



IBM

UniObjects for Java

Developer's Guide

Version 10.2
September, 2006

IBM Corporation
555 Bailey Avenue
San Jose, CA 95141

Licensed Materials – Property of IBM

© Copyright International Business Machines Corporation 2006. All rights reserved.

AIX, DB2, DB2 Universal Database, Distributed Relational Database Architecture, NUMA-Q, OS/2, OS/390, and OS/400, IBM Informix®, C-ISAM®, Foundation.2000™, IBM Informix® 4GL, IBM Informix® DataBlade® module, Client SDK™, Cloudscape™, Cloudsync™, IBM Informix® Connect, IBM Informix® Driver for JDBC, Dynamic Connect™, IBM Informix® Dynamic Scalable Architecture™ (DSA), IBM Informix® Dynamic Server™, IBM Informix® Enterprise Gateway Manager (Enterprise Gateway Manager), IBM Informix® Extended Parallel Server™, i.Financial Services™, J/Foundation™, MaxConnect™, Object Translator™, Red Brick® Decision Server™, IBM Informix® SE, IBM Informix® SQL, InformiXML™, RedBack®, SystemBuilder™, U2™, UniData®, UniVerse®, wIntegrate® are trademarks or registered trademarks of International Business Machines Corporation.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Windows, Windows NT, and Excel are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names used in this publication may be trademarks or service marks of others.

This product includes cryptographic software written by Eric Young (eay@cryptosoft.com).

This product includes software written by Tim Hudson (tjh@cryptosoft.com).

Documentation Team: Claire Gustafson, Shelley Thompson

US GOVERNMENT USERS RESTRICTED RIGHTS

Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Table of Contents

Preface

Organization of This Manual	ix
Documentation Conventions.	x
Help	xi
API Documentation	xii
Additional Reference	xiv

Chapter 1

Introduction

About UniObjects for Java	1-3
Benefits	1-4
Features	1-5

Chapter 2

Using UniObjects for Java

The Database Environment	2-4
Data Structure	2-4
File Dictionaries	2-5
Types of Dictionary Record	2-5
Locks	2-6
Data Retrieval	2-6
UniObjects Concepts	2-7
Objects	2-7
Methods	2-8
Properties	2-9
Opening a Database Session.	2-10
Using a Proxy Server	2-10
Data Encryption	2-10
Using Methods to Create Objects	2-11
Using the @TTY Variable.	2-11
Using Files	2-12
Reading and Writing Records	2-13

Fields, Values, and Subvalues	2-14
Data Conversion	2-16
Error Handling	2-17
Record Locks	2-18
Setting and Releasing Locks	2-18
Select Lists	2-20
Accessing Select Lists	2-20
Creating Select Lists	2-20
Reading and Clearing Select Lists	2-20
Using a Dictionary	2-21
Using Binary and Text Files	2-22
Accessing Files Sequentially	2-22
Using Database Commands	2-24
Client/Server Design Considerations	2-25
Calling Server Subroutines	2-25
When to Use Database Commands	2-26
Task Locks	2-26
Connection Pooling	2-27
Connection Pool Size	2-27
Connection Allocation	2-28
Activating Connection Pooling	2-28
Connection Pooling Program Example	2-30
UniObjects for Java Configuration File	2-34

Chapter 3 Using the Proxy Server

Setting Up the Proxy Server	3-3
Installing the Proxy Server Software	3-3
Editing the Proxy Server Configuration File	3-3
Starting the Proxy Server	3-7
Administering the Proxy Server	3-8
Getting Help for the Proxy Server	3-10

Chapter 4 A Tour of the Objects

Code Examples	4-5
Database Account Flavors	4-6
Objects and Methods Quick Reference	4-7
Unijava Methods	4-7
UniSession Methods	4-8
UniFile Methods	4-9
UniDictionary Methods	4-10

UniSequentialFile Object Methods	4-11
UniString Methods	4-12
UniDynArray Methods	4-13
UniRecord Methods	4-13
UniDataSet Methods	4-14
UniStringTokenizer Methods	4-15
UniSelect List Methods	4-15
UniCommand Methods	4-16
UniSubroutine Methods	4-16
UniTransaction Methods	4-17
UniNLSLocale Methods	4-17
UniObjects and BASIC Equivalents	4-18
UniJava Object	4-22
UniJava()	4-22
UniJava Object Methods	4-22
Example Using the UniJava Object	4-23
UniSession Object	4-25
UniSession Object Methods	4-26
Example Using the UniSession Object	4-49
UniFile Object	4-51
UniFile Object Methods	4-51
Example Using the UniFile Object	4-70
UniDictionary Object	4-71
UniDictionary Object Methods	4-71
Example Using the UniDictionary Object	4-95
UniSequentialFile Object	4-97
UniSequentialFile Object Methods	4-97
Example Using the UniSequentialFile Object	4-104
UniString Object	4-106
UniString Object Constructors	4-106
UniString Object Methods	4-107
UniStringTokenizer Object	4-117
UniStringTokenizer Object Constructors	4-117
UniDynArray Object	4-119
UniDynArray Object Constructors	4-119
UniDynArray Object Methods	4-120
Example Using the UniDynArray Object	4-125
UniRecord Object	4-126
UniRecord Object Constructors	4-126
UniRecord Object Methods	4-127

UniDataSet Object	4-129
UniDataSet Object Constructors	4-129
UniDataSet Object Methods	4-129
UniSelectList Object	4-137
UniSelectList Object Methods	4-137
Example Using the UniSelectList Object	4-142
UniCommand Object	4-144
UniCommand Object Methods	4-144
Example Using the UniCommand Object.	4-149
UniSubroutine Object	4-150
UniSubroutine Object Methods	4-150
Example Using the UniSubroutine Object	4-153
UniTransaction Object Methods	4-155
Example Using the UniTransaction Object	4-156
UniNLSLocale Object (UniVerse Only).	4-158
UniNLSLocale Object Methods.	4-158
UniNLSMap Object (UniVerse Only)	4-160
UniNLSMap Object Methods	4-160
UniException Object.	4-162
UniException Object Methods	4-162
UniXML Object	4-165
Public Methods	4-165

Chapter 5 Using SSL With UniObjects for Java

Overview of SSL Technology	5-3
Software Requirements	5-4
Setting up Java Secure Socket Extension (JSSE)	5-5
Configuring UOJ to use IBM JSSE	5-6
Configuring the Database Server for SSL	5-7
Creating a Secure Connection	5-9
Direct Connection	5-10
Establishing the Connection	5-12
Proxy Tunneling	5-13
Externally Secure	5-15
Managing Keys and Certificates for a UOJ Client and a Proxy Server	5-20
Importing CA Certificates Into UOJ Client Trustfile	5-20
Generating client certificates.	5-21
Managing Keyfile and Trustfile for the Proxy Server.	5-22

Appendix A Error Codes and Replace Tokens

Error Codes	A-2
@Variables	A-8
Blocking Strategy Values	A-9
Command Status Values	A-10
Host Type Values	A-11
Lock Status Values	A-12
Locking Strategy Values	A-13
fileSeek() Pointer Values	A-14
NLS Locale Values (UniVerse Only)	A-15
Release Strategy Values	A-16
System Delimiters	A-17
Encryption Values	A-18

Appendix B The Demo Application

Installing and Running the Demo	B-2
Installing the Demonstration Program	B-2
Running the Demonstration Program	B-2
Code Structure	B-4
Program Initialization	B-5
Declaring and Initializing Variables	B-5
Main Entry Point	B-5
Initializing FileDemo	B-6
Starting and Stopping FileDemo	B-8
Get Parent Window [Frame]	B-9
Controlling Data Output to Screen	B-9
Parent Frame	B-10
Accepting Logon Information for the Database	B-11
Connecting to the Database	B-13
Creating a Database File	B-15
Loading a Database File	B-17
Creating a Select List	B-18
Reading and Displaying the Select List	B-20
Listing the File	B-21
Executing a Command	B-23
Disconnecting from the Database.	B-26
Exiting FileDemo	B-27

Preface

This book describes UniObjects for Java, an interface to UniVerse and UniData databases from Java. The book is intended for experienced programmers and application developers who want to write Java programs that access UniVerse or UniData databases. The book assumes that you are familiar with UniVerse or UniData, and with Java. If you are new to Java, read one or more of the books in [“Additional Reference”](#) on page xiv. If you are new to UniVerse or UniData, you should read at least [The Database Environment](#) in Chapter 2, [“Using UniObjects for Java.”](#)

Organization of This Manual

This manual contains the following:

Chapter 1, [“Introduction,”](#) introduces UniObjects for Java and describes its benefits and features.

Chapter 2, [“Using UniObjects for Java,”](#) outlines the database environment and explains how to use UniObjects for Java to connect to the database, open files, access records, and so on.

Chapter 3, [“Using the Proxy Server,”](#) describes the proxy server and how to use it.

Chapter 4, [“A Tour of the Objects,”](#) is a guide to all the objects, methods and properties that are available with UniObjects for Java, and includes code examples for most objects.

Chapter 5, [“Using SSL With UniObjects for Java,”](#) discusses using the Secure Sockets Layer (SSL) with UniObjects for Java. SSL is a transport layer protocol that provides a secure channel between two communicating programs over which arbitrary application data can be sent securely.

Appendix A, [“Error Codes and Replace Tokens,”](#) lists error codes you may encounter when programming with UniObjects for Java.

Appendix B, [“The Demo Application,”](#) describes one of the demonstration applications supplied with UniObjects for Java.

Documentation Conventions

This manual uses the following conventions:

Convention	Usage
UPPERCASE	Uppercase indicates database commands, file names, keywords, BASIC statements and functions, and text that must be input exactly as shown.
<i>Italic</i>	Italic in a syntax line or an example indicates information that you supply. In text, words in italic are used for emphasis, or to reference a name, for example, an operating system path or a book title.
Courier	Courier indicates objects, methods, Java keywords, and examples of source code and system output.
This line â continues	The continuation character is used in source code examples to indicate a line that is too long to fit on the page, but must be entered as a single line on the screen.
[]	Brackets enclose optional items. Do not type the brackets unless indicated.

Documentation Conventions

The following conventions are also used:

- Syntax definitions and examples are indented for ease in reading.
- All punctuation marks included in the syntax—for example, commas, parentheses, or quotation marks—are required unless otherwise indicated.

Help

You can get Help about UniObjects for Java. In Windows Explorer, find and open the following file:

```
<Drive>:\IBM\UNIDK\uoj sdk\doc\asjava\tree.html
```

API Documentation

The following books document application programming interfaces (APIs) used for developing client applications that connect to UniVerse and UniData servers.

Administrative Supplement for Client APIs: Introduces IBM's seven common APIs, and provides important information that developers using any of the common APIs will need. It includes information about the UniRPC, the UCI Config Editor, the *ud_database* file, and device licensing.

UCI Developer's Guide: Describes how to use UCI (Uni Call Interface), an interface to UniVerse and UniData databases from C-based client programs. UCI uses ODBC-like function calls to execute SQL statements on local or remote UniVerse and UniData servers. This book is for experienced SQL programmers.

IBM JDBC Driver for UniData and UniVerse: Describes UniJDBC, an interface to UniData and UniVerse databases from JDBC applications. This book is for experienced programmers and application developers who are familiar with UniData and UniVerse, Java, JDBC, and who want to write JDBC applications that access these databases.

InterCall Developer's Guide: Describes how to use the InterCall API to access data on UniVerse and UniData systems from external programs. This book is for experienced programmers who are familiar with UniVerse or UniData.

UniObjects Developer's Guide: Describes UniObjects, an interface to UniVerse and UniData systems from Visual Basic. This book is for experienced programmers and application developers who are familiar with UniVerse or UniData, and with Visual Basic, and who want to write Visual Basic programs that access these databases.

UniObjects for Java Developer's Guide: Describes UniObjects for Java, an interface to UniVerse and UniData systems from Java. This book is for experienced programmers and application developers who are familiar with UniVerse or UniData, and with Java, and who want to write Java programs that access these databases.

UniObjects for .NET Developer's Guide: Describes UniObjects, an interface to UniVerse and UniData systems from .NET. This book is for experienced programmers and application developers who are familiar with UniVerse or UniData, and with .NET, and who want to write .NET programs that access these databases.

Using UniOLEDB: Describes how to use UniOLEDB, an interface to UniVerse and UniData systems for OLE DB consumers. This book is for experienced programmers and application developers who are familiar with UniVerse or UniData, and with OLE DB, and who want to write OLE DB programs that access these databases.

Additional Reference

Either of the following books may be useful if you are new to programming with Java:

Java in a Nutshell, 2nd Edition, by David Flanagan, May 1997
ISBN 1-56592-262-X, O'Reilly & Associates, Inc.

Core Java, 2nd Edition, by Gary Gornell & Cay S. Horstmann. ISBN 0-13-596891-7,
The Sunsoft Press.

Introduction

About UniObjects for Java.	1-3
Benefits	1-4
Features	1-5

This chapter provides an overview of UniObjects for Java and describes its benefits and features.

About UniObjects for Java

UniObjects for Java is a programming interface that lets developers create Java-based applications quickly and easily. It works with:

- UniVerse Release 9.4.1 or later on both Windows and UNIX servers
- UniData Release 5.1 or later on both Windows and UNIX servers

UniObjects for Java is a 100% Pure Java™ Class Library whose objects are able to take full advantage of any Java-based IDE (Integrated Development Environment).

UniObjects for Java provides the advantages of object-oriented development to Java-based client/server or Web development. On UniVerse systems, National Language Support (NLS) capabilities also provide developers with a unified method for creating applications usable in any country in the world. For complete information about UniVerse NLS, see the *UniVerse NLS Guide*.

In the case of Visual Cafe, UniObjects for Java becomes a natural extension of the development IDE. All objects can be manipulated in the same rapid and easy-to-use manner as a traditional UniVerse or UniData application.

UniObjects for Java supports rapid application development. Records are read and written using dynamic array objects (`UniDynArray` or `UniString`), which provide easy-to-use access to complex data structures representing real-world business objects such as orders, invoices, customers, and so on. Its ability to fully support the reuse of components also helps ensure the development of high quality applications faster, more efficiently, and often at lower cost.

Because of its capabilities and functionality, UniObjects for Java is the preferred API for Java-based development for UniVerse and UniData.

Benefits

UniObjects for Java provides the following benefits:

- UniObjects for Java supports the use of desktop development tools, including Visual Cafe, PowerJ, VisualJ++, and JBuilder, and uses native database Java objects for rapid application development (RAD) to create graphical Java applications quickly and easily.
- With Java IDEs, developers can select the best Java Beans, such as graphical or imaging add-ons, to use with UniObjects for Java. In this environment, each component can fully cooperate and exchange data seamlessly, enabling more functional applications.
- Developers can maximize their existing programming skills. This is especially true when using Visual Cafe or Visual J++ because UniObjects for Java provides desktop access to familiar database functions and syntax, while creating the latest well-partitioned applications.
- The result is reduced network traffic and increased overall maintainability over other client/server solutions.
- Companies can take advantage of Java technology for new GUI application interfaces. Yet they can also leverage today's database application by calling existing cataloged subroutines. Developers can easily change the front end without sacrificing years of investment in current technology.
- The ability to make secure connections between the UPJ client and a UniData or UniVerse server.

Features

UniObjects for Java features the following:

- 100% Pure Java™ Class Library, JDK 1.4 or higher
- Supports Visual Cafe, JBuilder, PowerJ, SuperCede, Visual J++, and other popular IDEs
- Provides a proxy server for additional Internet security and scalability
- Makes use of SSL (Secure Socket Layer) enabling secure connections from the client to the server
- Includes objects for files, sequential files, select lists, and so on
- Enables reuse of existing subroutines
- Offers direct mapping to database constructs and data models
- Provides integration with file dictionaries
- Allows application partitioning
- Features defined, consistent, and flexible locking strategies and exception handling
- Supports dynamic arrays and their inherent properties and methods
- Integrates with the database's transaction manager
- Is fully enabled for UniVerse NLS with appropriate character map and locale session properties

Using UniObjects for Java

The Database Environment	2-4
Data Structure	2-4
File Dictionaries	2-5
Types of Dictionary Record	2-5
Locks	2-6
Data Retrieval	2-6
UniObjects Concepts	2-7
Objects	2-7
Methods	2-8
Properties.	2-9
Opening a Database Session	2-10
Using a Proxy Server	2-10
Data Encryption	2-10
Using Methods to Create Objects	2-11
Using the @TTY Variable	2-11
Using Files	2-12
Reading and Writing Records	2-13
Fields, Values, and Subvalues	2-14
Data Conversion	2-16
Error Handling	2-17
Record Locks	2-18
Setting and Releasing Locks.	2-18
Select Lists	2-20
Accessing Select Lists.	2-20
Creating Select Lists	2-20
Reading and Clearing Select Lists	2-20
Using a Dictionary	2-21

Using Binary and Text Files	2-22
Accessing Files Sequentially.	2-22
Using Database Commands	2-24
Client/Server Design Considerations.	2-25
Calling Server Subroutines	2-25
When to Use Database Commands	2-26
Task Locks	2-26
Connection Pooling	2-27
Connection Pool Size	2-27
Connection Allocation.	2-28
Activating Connection Pooling	2-28
Connection Pooling Program Example	2-30
UniObjects for Java Configuration File	2-34

This chapter explains how to use UniVerse or UniData in a Java program. The topics covered include:

- An overview of the database environment
- Opening and controlling a database session
- Accessing files
- Locking records
- Handling errors
- Using dictionaries
- Accessing UniVerse text files and binary files for sequential processing
- Executing database commands
- Running subroutines on the server

If you are new to Java, you should read one of the books listed under [Additional Reference](#) in the “[Preface](#)” before you start this chapter.

For full details and more examples of all the objects and methods mentioned in this chapter, see Chapter 4, “[A Tour of the Objects.](#)”

The Database Environment

This section tells you just enough about the database environment to enable you to understand the rest of the chapter. If you already know about UniVerse or UniData, skip to “[UniObjects Concepts](#)” on page 2-7. To learn more about UniVerse, read *UniVerse System Description*. To learn more about UniData, read *Using UniData* and *Administering UniData*.

A database user logs on to a database account. A database account includes an operating system directory containing database files and possibly operating system files and directories too.



***Note:** UniVerse has several account flavors. The following sections describe the UniVerse IDEAL flavor, which is the recommended flavor to use with UniObjects. UniData uses ECLTYPE and BASICTYPE to specify account flavors. See Using UniData for information about UniData flavors.*

Each database file comprises a data file containing data records, and a file dictionary that defines the structure of the data records, how to display them, and so on. Each record in a file is uniquely identified by a record ID, which is stored separately from the data to which it refers.

The VOC file in a database account contains a record for every file used in the database. This record provides a cross-reference between the file name, which is the record ID, and the path of the file stored in field 2 of the record.

Data Structure

In an application each file holds one type of record. For example, a file called CUSTOMER might hold one record for each customer, whereas another file called ORDERS might hold one record for each order placed by a customer. The records and the fields they contain are not fixed in size, and the file itself can grow or shrink according to the amount of data it holds.

Data is stored in fields in a record. For example, a record in the CUSTOMER file might have fields containing the name, address, and telephone number of each customer. A field can hold more than one value, for example, the separate elements of an address can be stored as multivalued values of one field rather than as separate fields in the record. A field in one record can contain a cross-reference to data held in another file. For example, to link customers with their orders, records in the CUSTOMER file might have a multivalued field containing a list of the corresponding record IDs of their orders in the ORDERS file.

File Dictionaries

The file dictionary holds information about the structure of data records and their relationships to other files. In a record, each field is identified by a number, and the dictionary acts as a cross-reference between that number and the name of the field. For example, the customer's phone number might be held in a field called CUST.PHONE, which is field 3 in the record.

The file dictionary also defines how to format and display the data in the field for output, for example, the heading and the width of the column used in a report. All data is stored as character strings. Some data, such as monetary amounts and dates, is stored in a compact, internal format. For these fields the dictionary holds a conversion code, which specifies a conversion to apply before displaying the data.

Types of Dictionary Record

The following main types of dictionary record define fields in the data file:

- D-descriptors, which define the data actually stored in a field
- I-descriptors, which are calculated fields, evaluated whenever the value is required
- On UniData systems, V-descriptors (which define virtual fields) are like I-descriptors

I-descriptors can perform calculations on data stored in one record, or retrieve data from other files. For example, records in the CUSTOMER file have a field which lists related record IDs in the ORDERS file. The CUSTOMER file dictionary could contain I-descriptors that use the TRANS function to retrieve fields from those related records.

Locks

When a program makes changes to the database, it sets a lock on each record involved in the update, ensuring that no other user or process can modify the record until the lock is released. Locks and locking strategy are described in [“Record Locks”](#) on page 2-18.

Data Retrieval

UniVerse contains several utilities to use with the database, including:

- Retrieve, a data query and reporting language
- ReVise, a menu-based data entry and modification program
- Editor, a line editor that adds, changes, and deletes records in a file

UniData provides a set of similar programs:

- UniQuery, a data query and reporting language
- UniEntry, a menu-based data entry and modification program
- Editor and AE Editor, line editors that add, change, and delete records in a file
- UniData SQL, UniData’s version of the SQL language

The database also has many commands and keywords for administering and maintaining the database. All these utilities and commands can be accessed by a program through UniObjects. For more information, see [“Using Database Commands”](#) on page 2-23.

UniObjects Concepts

If you already know UniVerse or UniData, you will find that UniObjects uses some new terms to define familiar database features. This section defines those terms and shows how they map to the database.

Objects

An object is an instance of a class. All objects of a class share the same characteristics. The objects that you can use with UniObjects are shown in the following table.

Object	Description
UniJava	A UniJava object is the starting point for all applications and is used to access all UniSession objects.
UniSession	A UniSession object is a reference to a connection between your client program and the database running on the server. You normally access the other objects through the UniSession object.
UniFile	A UniFile object is a reference to a database file.
UniDictionary	A UniDictionary object is a reference to a database file dictionary.
UniDynArray	A UniDynArray object is a reference to a dynamic array, such as a record or select list.
UniSelectList	A UniSelectList object is a reference to a database select list.
UniDataSet	A UniDataSet object is a reference to a set of information, such as a group of record IDs, that can be used with other objects.
UniSequentialFile	A UniSequentialFile object is a reference to a type 1 or type 19 UniVerse file used for storing text, programs, or other data.

UniObjects for Java Objects

Object	Description
UniCommand	A UniCommand object is a reference to a database command executed on the server.
UniRecord	A UniRecord is a subclass of the UniDynArray class, and includes information regarding the record ID (as well as the record data) and the status of any recent operation on that data.
UniString	A UniString object is an abstraction of the standard <code>java.lang.String</code> class combined with the addition of functionality typically used by UniVerse and UniData developers.
UniStringTokenizer	A utility function that allows parsing of a given string. It is a modified version of the standard <code>java.util.StringTokenizer</code> class.
UniSubroutine	A UniSubroutine object is a reference to a BASIC subroutine that is called by the client program but runs on the server. For BASIC users, this is the familiar cataloged subroutine.
UniTransaction	A UniTransaction object is a reference to a transaction for a session.
UniNLSLocale	An UniNLSLocale object is a reference to the NLS locale information for a session.
UniNLSMap	An UniNLSMap object is a reference to the NLS map information for a session.

UniObjects for Java Objects (Continued)

Methods

Methods are procedures used with a particular object. Many of the methods used in UniObjects are equivalent to UniObjects methods and properties, and to BASIC statements and functions. For example, the `clearFile()` method is equivalent to the UniObjects **ClearFile** method and the BASIC CLEARFILE statement.

Properties

Properties represent the internal state of any given object. In UniObjects for Java you use get and set methods, such as `getFileName` and `setFileName`, to retrieve and change properties.

Opening a Database Session

You must connect to a database server before you can access files or records on it. You use the `connect ()` method of the `UniSession` object to establish a server session. The server can be the same computer that the client application is running on, or it can be a different computer linked by a network. A connected session is like any login session established by a terminal user.

Once the session is active, you can use it to create other objects. For example, if you want to open a file, execute a database command, or run a subroutine on the server, you start the operation using the methods provided by the `UniSession` object.

The `UniSession` object must exist for as long as your application needs access to the server. When a `UniSession` object is no longer active, the connection with the database server ends. This means that although the objects created through a `UniSession` object are still available, you may not be able to use them. For example, if you have a `UniFile` object, you can access the last record read from the file, but you cannot read another record.

Using a Proxy Server

The `UniObjects for Java proxy server` lets users of applets access data sources on hosts other than the one serving the applet. It also lets users access data sources protected by a network firewall, providing secure access through the firewall.

When a client application uses the proxy server, it sends a request to the proxy server, which then routes the request to the server. The server then responds to the proxy server, which in turn sends the response back to the client.

Data Encryption

Data encryption is a facility in which data transmissions between the client and server is modified to prevent unsecure parties from intercepting sensitive data. `UniObjects for Java` provides the facility to use encryption at the session, object, and operation levels.

Using Methods to Create Objects

The following table shows the UniObjects for Java objects and the methods you use to create or access them. The methods all belong to the UniSession object unless otherwise stated.

Object	Method
UniFile	open() method.
UniDictionary	openDict() method.
UniSequentialFile	openSeq() method.
UniDynArray	dynArray() method of UniSession object, or readList() method of UniSelectList object. Can also be created independently.
UniSelectList	selectList() method.
UniCommand	command() method.
UniSubroutine	subroutine() method.
UniTransaction	starttransaction() method.
UniNLSLocale	nlsLocale() method.
UniNLSMap	nlsMap() method.

UniObjects for Java Objects and Methods

Using the @TTY Variable

During normal server operations, the @TTY variable on the server is set to the terminal number. If the process is a phantom, @TTY returns the value phantom. If the process is a database API such as UniObjects for Java or UniObjects, @TTY returns the value uvcs on UniVerse systems and udcs on UniData systems.

You can use this returned value by adding a paragraph entry to the VOC file. For example:

```
PA
IF @TTY = 'uvcs' THEN GO END:
START.APP
END:
```

Using Files

Before you can use a database file, you must open the file using the `open()` method of the `UniSession` object as follows:

```
UniFile custfile = uSession.open("CUST");
```

Reading and Writing Records

Once a file is open, you can read data from it and write data to it. To read a record, call the `read()` method of the `UniFile` object. For example:

```
UniString custRec = custFile.read("12345");
```

To write data back to the file, call the `write()` method of the `UniFile` object. For example:

```
custFile.write();
```

Fields, Values, and Subvalues

When you read a record, it is returned as a `UniDynArray` object. This object is an extension of the `UniString` object, meaning that all methods available to the `UniString` objects are also available to the `UniDynArray` object. To access or manipulate the dynamic array, you address the fields, values, and subvalues just as you do in BASIC and just as they are stored in the file.

You can address fields, values, and subvalues as follows:

```
dynArray.method (field, [value, [subvalue]]);
```

`dynArray` is the object variable, `method` is the method you want to use, and `field`, `value`, and `subvalue` are integers representing the respective field, value, or subvalue you want to access. If no field is given, the operation occurs over the entire array.

For example, to find what is the third value in a dynamic array, write:

```
UniDynArray thirdField = origArray.extract( 3 );
```

This extracts field 3 from the `origArray` and returns the data into the object `thirdField`.

To access a value, do the same thing, but extend it as follows:

```
UniDynArray thirdFieldSecondValue = origArray.extract( 3, 2 );
```

This extracts the second value from the third field and returns the object into `thirdFieldSecondValue`.

To modify data in the object, do the same thing. For example, to change the second value of the third field, write:

```
origArray.replace( 3, 2, "NewData" );
```

This changes the object immediately.

Other operations you can perform are:

- To count the number of values in field 2 of the dynamic array:

```
int NumValues = origArray.dcount( 2 );
```

- To count the number of fields in the entire array:

```
int NumValues = origArray.dcount();
```

- To insert a new field before field 5 in the array:

```
origArray.insert( 5, "new value " );
```

- And finally, to delete the fourth subvalue of the first value of field 3:

```
origArray.delete( 3, 1, 4 );
```

You can also use the other methods in the same way.

Data Conversion

When you read and write an entire record, your program must handle conversion of data to and from its internal storage format. You do this using the `iconv()` (input convert) and `oconv()` (output convert) methods of the `UniSession` object.

For example:

```
UniString dateBox = uSession.oconv(x, "D");
```

In most cases the position of the field in the record and the conversion code to apply must be written into your program. This means that your program may need to change if the structure of the record changes.

As an alternative, you can read or write to a named field rather than to the entire record, and let `UniObjects` consult the file dictionary and perform any data conversion for you. You do this using the `readNamedField()` and `writeNamedField()` methods.¹

The `readNamedField()` method of the `UniFile` object lets a program request data in its converted form from a field specified by name. `readNamedField()` can also evaluate I-descriptors. For example, the code to read the `LAST.ORDER.DATE` field might look like this:

```
UniString rec = custFile.readNamedField('LAST.ORDER.DATE');
```

The `writeNamedField()` method does the converse, that is, it takes a data value, applies an input conversion to it, then writes it to the appropriate location in the record. It does not support I-descriptors.

1. BASIC does not have equivalents to the `readNamedField()` and `writeNamedField()` methods.

Error Handling

UniObjects for Java separates the code that handles error exceptions from the normal code flow. An exceptional condition is said to *throw an exception* that must be *caught*. Whenever an error occurs in one of the API libraries, the method encountering the error throws a particular exception, which the programmer must then catch and handle appropriately. This is done using Java *try/catch* blocks. For example:

```
try
{
    result = uFile.read(recordToBeRead);
    result2 = uFile.read(nextRecordToBeRead);
}
catch(UniException e)
{
    processError(e);
}
```

This ensures that normal operations are handled in one section, and exceptional conditions are handled in another section.

Many classes support a `status()` method, which lets the developer get additional information about certain operations.

UniObjects for Java does not have a direct equivalent to the THEN and ELSE clauses that a BASIC programmer uses to specify different actions depending on the success of an operation. Instead, all database objects throw a `UniException` object, which is set by various methods. If the method does not finish successfully, the `UniException` object indicates an error. For a list of error codes, see Appendix A, [“Error Codes and Replace Tokens.”](#)

For example, if you call the `read()` method of a `UniFile` object, the operation fails if the record does not exist. In this case, the `UniFileException` object indicates the record was not found.

For examples of error handling, see the entry for the `UniFile` object in Chapter 4, [“A Tour of the Objects.”](#)



Record Locks

Note: BASIC programmers should read this section carefully. Locking is handled differently in UniObjects to make coding easier in the event-driven environment of a client application.

UniVerse and UniData have a system of locks to prevent potential problems when several users try to access the same data at the same time. The three types of lock you can use in programs are task locks, file locks, and record locks. This section discusses only record locks, which are used most often. For information on task locks and file locks, refer to the descriptions of the `setTaskLock()` and `releaseTaskLock()` methods of the `UniSession` object, and to the `lockFile()` and `unlockFile()` methods of the `UniFile` object, in Chapter 4, “A Tour of the Objects.” Also see *UniVerse System Description* for more information on record and file locks.

A record lock prevents other users from:

- Setting a file lock on the file containing the locked record.
- Setting a record lock on the locked record.
- Writing to the locked record.
- Creating a record with the same record ID. In this case you set a lock on the record before it has been created.

There are two types of record lock:

- Exclusive update locks (READU locks), which prevent other users from reading or writing to the record
- Shared read locks (READL locks), which allow other users to read the record but not to update it

Setting and Releasing Locks

Setting and releasing record locks is controlled by three methods of the `UniFile` or `UniDictionary` object:

- The blocking strategy set by the `setDefaultBlockingStrategy()` method specifies whether to wait if the record is already locked (equivalent to a BASIC LOCKED clause).



- The lock strategy set by the `setDefaultLockStrategy()` method specifies what kind of lock to set when reading.
- The release strategy set by the `setDefaultReleaseStrategy()` specifies when to release a lock, for example:
 - When a record is written or deleted.
 - When you set a new record ID value using the `setRecordID()` method. This provides a simple way to set the lock release strategy for a program that edits a sequence of records, without having to code lock handling every time a record is read or written.
 - Only by the `unlockRecord()` method.

Note: All locks are released when the session is closed.

You can set these properties for each file, or you can use the defaults associated with the `UniSession` object. These defaults are specified using the `setDefaultBlockingStrategy()`, `setDefaultLockStrategy()`, and `setDefaultReleaseStrategy()` methods of the `UniSession` object. In either case the properties remain set for all subsequent reads on that file during the session; you do not need to set them again. For examples, see the entries for the `UniFile` and `UniSession` objects in Chapter 4, “[A Tour of the Objects.](#)”

Select Lists

In UniVerse and UniData you can retrieve a specified set of record IDs, saving them as an active select list. You can either use the active select list immediately in a program or command, or give it a name and save it for future use. A UniVerse session can have up to 11 select lists active at the same time, numbered from 0 through 10. A UniData session can have up to 10 active select lists, numbered from 0 through 9.

Accessing Select Lists

A UniObjects for Java program can use select lists by defining `UniSelectList` objects. You get a reference to one of the numbered select lists using the `selectList()` method of the `UniSession` object, for example:

```
UniSelectList uSelect = uSession.selectList(1);
```

Creating Select Lists

The `UniSelectList` methods you can use to create a select list are `formList()`, `select()`, `selectAlternateKey()`, or `selectMatchingAk()`. You can also create a select list by executing a database command that creates one, for example, `SELECT` or `SSELECT`. In the following example the `select()` method creates a select list:

```
uSelect.select(uFile);
```

Reading and Clearing Select Lists

You can read a select list in two ways:

- One record ID at a time using the `next()` method
- All record IDs at once using the `readList()` method

If you just want to read part of a list, you can discard the unwanted part by calling the `clearList()` method.

For more information and examples, see the entry for the `UniSelectList` object in Chapter 4, “[A Tour of the Objects.](#)”

Using a Dictionary

For most application programs it is economical to build a record's structure and field types into the program. This avoids having to look up the format of the record in the file dictionary. If you want your program to process different types of records, you will need to look in the file dictionary to see how the records are structured. In a UniObjects for Java program you do this through the `openDict()` method of the `UniSession` object. This returns a `UniDictionary` object, which has methods for reading and writing particular fields from the dictionary. These methods are:

- `getAssoc()` and `setAssoc()`
- `getConv()` and `setConv()`
- `getFormat()` and `setFormat()`
- `getLoc()` and `setLoc()`
- `getName()` and `setName()`
- `getSM()` and `setSM()`
- `getSQLType()` and `setSQLType()`
- `getType()` and `setType()`

For more information about these methods, see the entry for the `UniDictionary` object in Chapter 4, [“A Tour of the Objects.”](#)

Here is an example that finds the type of a particular field:

```
UniDictionary dictFile = uSession.openDict ("XXX")
UniString rec = dictFile.getType();
```

Using Binary and Text Files

You can use UniVerse type 1 and type 19 files to store text or binary data you want to include in a program. UniVerse implements type 1 and type 19 files as operating system directories. The records in type 1 and type 19 files are implemented as operating system files whose filenames are the database record IDs:

Database Item	Implemented by Operating System as...
Type 1 or type 19 file	Directory
Type 1 or type 19 file record	File
Record ID	Filename

For small text files, you can open the type 1 or type 19 file with the `open()` method and then read an entire text file with the `read()` method. See [“Using Files”](#) on page 2-12.

Accessing Files Sequentially

On UniVerse and UniData systems, if a file is large or contains binary data, it is better to read and write the file sequentially, that is, in manageable sections. You can do this by using the `openSeq()` method of the `UniSession` object. This returns a `UniSequentialFile` object, whose methods allow sequential access to the data. The `UniSequentialFile` object uses an internal file pointer to track read and write operations (equivalent to BASIC’s sequential file variable). You can:

- Read and write lines of text with the `readLine()` and `writeLine()` methods
- Read and write binary data with the `readBlk()` and `writeBlk()` methods
- Change the position of the file pointer with the `fileSeek()` method
- Truncate an existing file with the `writeEOF()` method

For more information, see the entry for the `UniSequentialFile` object in Chapter 4, [“A Tour of the Objects.”](#)

Using Database Commands

Your program can run most database commands through the `UniCommand` object, which is equivalent to the BASIC EXECUTE statement.

You can use the `UniCommand` object for:

- Creating or deleting a database file.
- Making a select list of records that meet your requirements. See [“When to Use Database Commands”](#) on page 2-25.
- Running a program on the server to save processing power on the client.

***Note:** The `UniCommand` object may not always be the most efficient way to use resources in a client/server program. For more information, see [“When to Use Database Commands”](#) on page 2-25.*

You can issue only one command at a time. You use the `command()` method of the `UniSession` object to create a `UniCommand` object. For example:

```
UniCommand com1 = uSession.command()
```

You specify the command that you want to execute using the `setCommand()` method, then execute it by calling the `exec()` method. For example:

```
com1.setCommand("some command");  
com1.exec();
```

You can get the result of a command using the `getCommandStatus()` and `response()` methods as follows:

- If the command ran to completion, `getCommandStatus()` returns `UniObjectsTokens.UVS_COMPLETE`, and you can get any output generated by the command using the `response()` method.
- If the command did not finish, or if all the output was not retrieved, the `getCommandStatus()` method shows what happened. You can use the `reply()` or `nextBlock()` method to continue processing. For an example, see the entry for the `UniCommand` object in Chapter 4, [“A Tour of the Objects.”](#)



Client/Server Design Considerations

In designing your application, avoid unnecessary interaction between the client and the server. This has two main benefits:

- Performance: reducing network traffic improves performance.
- Scalability: if more clients and servers are added to the network, your application's performance remains acceptable.

To use the client and server efficiently, you must know which operations need to communicate with the server and when those operations take place. If necessary, you can then change the design of the application to reduce the interaction with the server. The following sections describe some ideas for using the client and server economically.

Calling Server Subroutines

You can reduce network traffic by running parts of your application on the server as BASIC subroutines. Server subroutines run in an area called catalog space that is available to any program on the server.



***Note:** A server subroutine must be cataloged before you can call it from UniObjects. For more information about cataloging subroutines on UniVerse systems, see the entry for the CATALOG command in UniVerse User Reference, and the discussion of subroutines in UniVerse BASIC. For information about cataloging UniData subroutines, see UniData Commands Reference and Administering UniData.*

Your program can call a cataloged subroutine through the UniSubroutine object, which you get using the subroutine () method of the UniSession object. For example:

```
UniSubroutine getOrderData = uSession.subroutine  
("GET.ORDER.DATA", 4);
```

The `subroutine()` method needs the name of the cataloged subroutine and the number of arguments that it takes. Once your program has obtained the `UniSubroutine` object, you use the `setArg()` method to supply values for arguments, the `call()` method to call the subroutine, and the `getArg()` method to retrieve any argument values returned. For example:

```
getOrderData.setArg(0, OrderNumber);
getOrderData.setArg(1, DisplayType);
getOrderData.call();
UniString displayValue = getOrderData.getArg(2);
```

When to Use Database Commands

You can save client processing by executing database commands on the server. The most effective commands to use are those that do not generate any output, such as the `Retrieve` and `UniQuery SELECT` and `SSELECT` commands.

Some commands can increase network traffic because they generate prompts or messages that your program must then handle. If your program cannot cope with an unexpected request for input from a command, it hangs, with no indication of what went wrong. In particular, avoid using interactive commands such as `CREATE.FILE` or `REFORMAT` which have many possible prompts and error conditions. (In most cases it should not be necessary to create or reformat files as part of your application.)

Task Locks

You can protect a process running on the server from interruption by other users or programs by setting a task lock. `UniVerse` and `UniData` have 64 task locks you can assign to events or processes. For example, if your application uses a resource such as a printer, you can set a task lock to prevent another database user from accessing the printer during your print run.

You set and release task locks with the `setTaskLock()` and `releaseTaskLock()` methods of the `UniSession` object. Task locks have no predefined meanings. You must ensure that your application sets and releases task locks efficiently. You can use the `LIST.LOCKS` command to check which locks are in use and which users hold them.

Connection Pooling

UniData 7.1 supports connection pooling with UniObjects for Java and UniObjects for .NET.

The term *connection pooling* refers to the technology that pools permanent connections to data sources for multiple threads to share. It improves application performance by saving the overhead of making a fresh connection each time one is required. Instead of physically terminating a connection when it is no longer needed, connections are returned to the pool and an available connection is given to the next thread with the same credentials.

You can activate connection pooling in your program, or activate it through a configuration file.

Connection Pool Size

You can set the minimum and maximum size of the connection pool either in your program or through a configuration file. If you do not define these sizes, the minimum size defaults to 1 and the maximum size defaults to 10. The minimum size determines the initial size of the connection pool.

The size of the connection pool changes dynamically between the minimum and maximum sizes you specify, depending on the system demands. When there are no pooled connections available, UniData either creates another connection, if the maximum connection pool size has not been reached, or keeps the thread waiting in the queue until a pooled connection is released or the request times out. If a pooled connection is idle for a specified time, it is disconnected.

License Considerations

The actual size of a connection pool depends on the pooling licenses available on the server. For example, if you set a connection pool to a minimum size of 2 and a maximum size of 100, and you have 16 licenses available, the maximum connection pool size will be 16. If you only have 1 license available, UniData does not create the connection pool at all, since the minimum size of 2 cannot be met.

Connection Allocation

Once UniData allocates a pooled connection to a thread, the connection remains exclusively attached to that thread until it is explicitly freed by the thread.

UniData does not “clean up” the pooled connection before allocating it to a user thread with the same credentials. For example, UDT.OPTIONS settings, unnamed common, environment variables, and so forth remain from previous use.

Activating Connection Pooling

To activate connection pooling, use the `UniObjects.UOPooling` statement in your program, as shown in the following example.,

```
UniObjects.UOPooling = true;
```

Specifying the Size of the Connection Pool

To specify the size of the connection pool, use `UniObjects.MinPoolSize` to define the minimum number of connections, and the `UniObjects.MaxPoolSize` to define the maximum number of connections, as shown in the following example:

```
UniObjects.MinPoolSize = 1;  
UniObjects.MaxPoolSize = 10;
```

If you do not specify the minimum and maximum number of connections, UniData defaults to 1 for the minimum and 10 for the maximum.

Creating Multiple Connection Pools

You can create as many connection pools as you like by issuing multiple `UniObjects.OpenSession` commands in your program. You must specify different credentials for each connection pool.

```
UniObjects.OpenSession(server_name, logon_name,  
password,account,service_name)
```

The following table describes each parameter of the syntax.

Parameter	Description
<i>server_name</i>	The name of the server to which you are connecting.
<i>logon_name</i>	The logon name of the user connecting to the server.
<i>password</i>	The password corresponding to the <i>logon_name</i> .
<i>account</i>	The account on the server to which you are connecting.
<i>service_name</i>	This parameter is optional. The name of the rpc service. If you do not specify <i>service_name</i> , UniData defaults to defcs. If you do specify <i>service_name</i> , the service name must exist in the unirpcservices file.

UniObjects.OpenSession Parameters

When you close a session using connection pooling, UniData does not close the connection, it makes the connection available in the connection pool.

Connection Pooling Program Example

The following example illustrates using connection pooling in a program for UniObjects for Java:

```
/*
 * Created on Jun 15, 2005
 *
 * To change the template for this generated file go to
 * Window>Preferences>Java>Code Generation>Code and Comments
 */
package asjava.uniCPTest;
import java.io.IOException;
import java.util.Date;
import asjava.uniclientlibs.*;
import asjava.uniobjects.*;
/**
 * @author nikk
 *
 * To change the template for this generated type comment go to
 * Window>Preferences>Java>Code Generation>Code and Comments
 */
public class CPTest {
    CPTest() {
    };
    static Thread[] threads = null;
    static int m_NumberOfThreads = 0;
    static String numberOfThreads;
    static String _hostName;
    static String _userName;
    static String _passWord;
    static String _accountPath;
    static String _dataSourceType = "Unknown";
    static boolean goodResponse = false;
    static String response = null;
    static String _minPoolSize;
    static String _maxPoolSize;

    public static void main(String args[]) throws IOException, Throwable {
        if (_hostName == null) {
            _hostName = inputString("Host Name:");
        }
        if (_userName == null) {
            _userName = inputString("User Name:");
        }
        if (_passWord == null) {
            _passWord = inputString("User Password:");
        }
        if (_dataSourceType.equals("Unknown")) {
            goodResponse = false;
            while (goodResponse == false) {
                response =
                    inputString("Is Server UniVerse" + " or UniData (V/D):");
                if (response.equalsIgnoreCase("V")) {
                    _dataSourceType = "UNIVERSE";
                    goodResponse = true;
                } else if (response.equalsIgnoreCase("D")) {
                    _dataSourceType = "UNIDATA";
                    goodResponse = true;
                }
            }
        }
    }
}
```

```

        } else {
            System.out.println("... Enter V or D");
        }
    }
}
if (_accountPath == null) {
    _accountPath = inputString("Account Path:");
}
if (_minPoolSize == null) {
    _minPoolSize = inputString("Min Pool Size:");
}
if (_maxPoolSize == null) {
    _maxPoolSize = inputString("Max Pool Size:");
}
if (numberOfThreads == null) {
    numberOfThreads =
        inputString("Number of Simultaneous Connections : ");
}
m_NumberOfThreads = Integer.parseInt(numberOfThreads);
long lStartTime = new Date().getTime();
ConcurrentTest2();
for (int i = 0; i < m_NumberOfThreads; i++) {
    threads[i].join();
}
// Console.WriteLine("Program Finished" + DateTime.Now);
long ldiff = (new Date().getTime() - lStartTime);
System.out.println("Total Time : " + ldiff + " Milliseconds");
System.in.read();
}
public static String inputString(String msg) throws IOException {
    String userInput;
    byte bArray[] = new byte[128];
    int bytesRead;
    System.out.print(msg);
    bytesRead = System.in.read(bArray);
    userInput = new String(bArray, 0, bytesRead);
    // remove trailing CR/LF and any other leading or trailing white
    // space
    userInput = userInput.trim();
    return (userInput);
}
private static void ConcurrentTest2() {
    int nIndex = m_NumberOfThreads;
    threads = new Thread[nIndex];
    for (int i = 0; i < nIndex; i++) {
        Thread t = new Thread(new ThreadStart());
        threads[i] = t;
    }
    for (int i = 0; i < nIndex; i++) {
        threads[i].setName("MyThreadProc" + (i + 1));
        threads[i].start();
    }
}
static class ThreadStart implements Runnable {
    UniSession us1 = null;
    UniJava j = null;
    public void run() {
        try {
            // UniJava.setIdleRemoveThreshold(20000);
            // UniJava.setIdleRemoveExecInterval(20000);

```

```

        j = new UniJava();
        //    you can comment out below lines and set the Connection
Pooling using
        //    Configuration file called "uoj.properties". This way you
do not have to
        //    change the code. Put "uoj.properties" file in current
working directory.
        //    Installation contains "uoj.properties" file.
UniJava.setUOPooling(true);
Integer minSize = Integer.decode(_minPoolSize);
Integer maxSize = Integer.decode(_maxPoolSize);
UniJava.setMinPoolSize(minSize.intValue());
UniJava.setMaxPoolSize(maxSize.intValue());
//    it will create trace file (uoj_trace.log)in current
working directory
UniJava.setPoolingDebug(true);
us1 = j.openSession();
us1.setHostName(_hostName);
us1.setUserName(_userName);
us1.setPassword(_passWord);
us1.setAccountPath(_accountPath);
if (_dataSourceType.equals("UNIDATA")) {
    us1.setConnectionString("udcs");
} else {
    us1.setConnectionString("uvcs");
}
us1.connect();
UniCommand cmd = us1.command();
cmd.setCommand("SSELECT STATES");
cmd.exec();
String s = cmd.response();
System.out.println(" Response from UniCommand : " + s);
UniSelectList sl = us1.selectList(0);
try {
    String sFM = us1.getMarkCharacter(UniTokens.FM);
    int bb = 0;
} catch (UniConnectionException e1) {
    //    TODO Auto-generated catch block
    e1.printStackTrace();
}
while (!sl.isLastRecordRead()) {
    UniString s2 = sl.next();
    if(!s2.equals(""))
    {
        System.out.println(" Record ID : " + s2);
    }
}
} catch (UniSessionException ex) {
    if (us1 != null && us1.isActive()) {

        try {
            j.closeSession(us1);
        } catch (UniSessionException e) {
            //    TODO Auto-generated catch block
            e.printStackTrace();
        }
        us1 = null;
    }
    ex.printStackTrace();
    System.out.println(ex.getMessage());
} catch (UniCommandException e) {

```

```

        //      TODO Auto-generated catch block
        e.printStackTrace();
    } catch (UniSelectListException e) {
        //      TODO Auto-generated catch block
        e.printStackTrace();
    } finally {
        if (us1 != null && us1.isActive()) {
            System.out.println("");
            System.out.println(
                Thread.currentThread().getName()
                    + " : Connection Passed in Test Program");
            System.out.println(
                "=====
                ==");
                System.out.println(
                    "=====
                    ==");
                try {
                    j.closeSession(us1);
                } catch (UniSessionException e) {
                    //      TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }
    }

```

UniObjects for Java Configuration File

You can set parameters for connection pooling in a the `uoj.properties` configuration file residing on the client in the current directory. The following example illustrates the `uoj.properties` file:

```
ConnectionPoolingOn=1
PoolingDebug=1
MinimumPoolSize=1
MaximumPoolSize=16
IdleRemoveThreshold=300000
IdleRemoveExecInterval=60000
OpenSessionTimeOut=30000
```

The following table describes each parameter in the `uoj.properties` configuration file:

Parameter	Description
ConnectionPoolingOn	When this value is set to 1, the connection is drawn from the appropriate pool, or, if necessary, created and added to the appropriate pool. The default value is 0.
PoolingDebug	When this value is set to 1, UniObjects for Java creates the <code>uoj_trace.log</code> file in the current directory for use in debugging and tracing.
MinimumPoolSize	The minimum number of connections maintained in the connection pool.
MaximumPoolSize	The maximum number of connections in the connection pool.
IdleRemoveThreshold	Determines the amount of time, in milliseconds, one session can remain idle in the connection pool.
IdleRemoveExecInterval	The thread execution interval time, in milliseconds. During this interval, UniObjects for Java removes idle sessions in the connection pool.
OpenSessionTimeOut	How much time UniObjects for Java waits before timing out to get a session from the connection pool. Expressed in milliseconds.

uoj.properties Parameters

Using the Proxy Server

Setting Up the Proxy Server	3-3
Installing the Proxy Server Software	3-3
Editing the Proxy Server Configuration File	3-3
Starting the Proxy Server	3-7
Administering the Proxy Server	3-8
Getting Help for the Proxy Server	3-10

The UniObjects for Java proxy server serves two general purposes:

- It lets users of applets access data sources on hosts other than the one serving the applet.
- It lets users access data sources protected by a network firewall, providing secure access through the firewall.

UniObjects for Java can be used in applets and applications. But if an applet is loaded from a remote host, the host machine serving the applet is the only machine to which the applet can open a network connection. Since a data server is almost always a different machine from the web server, the UniObjects for Java proxy server is required to overcome this security restriction.

In network environments where a firewall protects access to a company intranet, no unauthorized network traffic can occur to systems behind the firewall. The data server in such cases does not reside on the same system as the firewall, so the proxy server is needed in these cases, too.

The proxy server resides:

- On the same host as the one that serves as the network firewall
- On the web server that serves the UniObjects for Java applet

The proxy server routes the RPC traffic created by the client and the server. It also provides authentication capabilities for both client and server, ensuring a degree of security, especially for networks using a firewall.

When a client application uses the proxy server, it sends a request to the proxy server, which then routes the request to the server. The server then responds to the proxy server, which in turn sends the response back to the client. All client requests are synchronous.

The UniObjects for Java proxy server allows a client application to communicate with multiple servers, too. Because all client requests are synchronous, the proxy server does not need to synchronize responses from multiple data sources.

Setting Up the Proxy Server

Before you can use a server system as a proxy server, you must do the following:

- Install the proxy server software on the system you are setting up.
- Edit the proxy server configuration file *uniproxy.config*.

Installing the Proxy Server Software

1. On the proxy server system, install a Java development kit (JDK), version 1.4 or higher.
2. Copy the following two files from a database server to the proxy server system:
 - On UNIX systems:
 - *unishared/uojsdk/lib/asjava_p.zip*
 - *unishared/uojsdk/lib/asjava.zip*
 - On Windows platforms:
 - `<unidkhome>\uojsdk\lib\asjava_p.zip`
 - `<unidkhome>\uojsdk\lib\asjava.zip`

You can put these files anywhere on the proxy server system.
3. Set the CLASSPATH environment variable to include the full paths of all the following files:
 - *classes.zip* (or whatever the JDK archive is called)
 - *asjava_p.zip*
 - *asjava.zip*

Editing the Proxy Server Configuration File

Use any editor to edit the *uniproxy.config* file. The following settings are included in the default *uniproxy.config* file:

The TCP/IP port that the proxy server accepts incoming connection requests on:

```
PROXY_PORT=31448
```

The TCP/IP port that the proxy server accepts incoming administrative commands on:

```
ADMIN_PORT=31458
```

The password that must be included with any administrative commands submitted to the proxy server:

```
ADMIN_ACCESS_TOKEN=password9
```

The I/O stream buffer size. This is the largest unit of data that the proxy server can transport for a single connection:

```
BUFFER_SIZE=4096
```

The debug level controls the amount of output that is written to the logging file. 0 means only errors are written out. 9 means every data packet is logged. The levels between 0 and 9 let you log information between those extremes.

```
DEBUG_LEVEL=9
```

The maximum number of client connections allowed:

```
MAX_CONNECTIONS=12
```

The maximum number of server connections allowed per client connection:

```
MAX_MULTIPLEXED_SERVERS=12
```

The name of the log file to use:

```
NAME_LOG=testLog
```

The time in milliseconds that the proxy server waits before disconnecting an inactive connection:

```
NETWORK_TIMEOUT=120000
```

The O/S-specific file system path under which you want to store your log file. Make sure to include double backslashes in your paths and end them with a slash:

```
PATH_LOG="D:\\JDK1.4\\Projects\\unijava\\uniproxy\\"
```

Proxy Server Access List

The following rules dictate how access can be granted or denied to the proxy server:

1. If you define ACCESS_TOKENs only, any one providing one of those tokens will have full access to all systems through this proxy server.

2. If you define one or more `ACCESS_SERVER` tokens, users will have access to only those servers that have `ACCESS_SERVER` entries.
3. If you define one or more `ACCESS_TOKEN_SERVER` tokens immediately following an `ACCESS_TOKEN`, that `ACCESS_TOKEN` provides access only to those servers that have `ACCESS_TOKEN_SERVER` entries.

```
ACCESS_TOKEN=password1
ACCESS_TOKEN_SERVER=mickey
ACCESS_TOKEN_SERVER=mickey
ACCESS_TOKEN_SERVER=goofy
ACCESS_TOKEN_SERVER=goofy
ACCESS_TOKEN_SERVER=localhost
ACCESS_TOKEN=password2
ACCESS_SERVER=goofy
ACCESS_SERVER=goofy
ACCESS_SERVER=mickey
ACCESS_SERVER=mickey
ACCESS_SERVER=localhost
```

You must set the following parameters on your proxy server system:

- `ADMIN_ACCESS`. This is the administrator's password to the proxy server.
- `DEBUG_LEVEL`. Normally you set the debug level to 0.
- `MAX_CONNECTIONS`. Normally you set this to the maximum number of licenses on the database server.
- `MAX_MULTIPLEXED_SERVERS`. We recommend you set this to at least two times the number of servers protected by the proxy server. Increase this number as you need to.
- `PATH_LOG`. This parameter defines the path of the proxy server log directory.
- `ACCESS_TOKEN`, `ACCESS_SERVER`, and `ACCESS_TOKEN_SERVER`. These are passwords giving different kinds of access to servers.

Starting the Proxy Server

Use the **java asjava.uniproxy.UniProxyAdminClient** command to start the proxy server. Its syntax is as follows:

```
java asjava.uniproxy.UniProxyAdminClient config=path
command=start [ access_token=token ]
```

The following table describes each parameter of the syntax.

Parameter	Description
asjava.uniproxy.UniProxyAdminClient	Starts an instance of the proxy administration client in a Java run time, and passes the rest of the command-line arguments to the client.
-config= <i>path</i>	Specifies the path of the proxy configuration file.
-command=start	Starts the proxy server.
-access_token= <i>token</i>	Verifies that the user running the <i>java</i> command has administrative access to the proxy server.

java asjava.uniproxy.UniProxyAdminClient Parameters

Administering the Proxy Server

Use the **java asjava.uniproxy.UniProxyAdminClient** command to administer the proxy server. With this command you can do the following:

- Start, suspend, shut down, and restart the proxy server.
- Reload an updated proxy server configuration file.
- Write the current status of the proxy server to the log file.

The syntax is as follows:

```
java asjava.uniproxy.UniProxyAdminClient port=port  
command=command [ server=name ] [ access_token=token ]
```

Parameter	Description
asjava.uniproxy.UniProxyAdminClient	Starts an instance of the proxy administration client in a Java run time, and passes the rest of the command-line arguments to the client.
-port= <i>port</i>	Specifies the port number on which the proxy server is running.
-command= <i>command</i>	Specifies the command to run on the proxy server. The next section lists all the commands you can use.
-server= <i>name</i>	Specifies the name of the machine the proxy server is running on.
-access_token= <i>token</i>	Verifies that the user running the <i>java</i> command has administrative access to the proxy server.

java asjava.uniproxy.UniProxyAdminClient Parameters

Proxy Server Commands

Using the *command* option, the *java* command can execute any of the following proxy server commands:

Command	Description
start	Starts a new instance of the proxy server. This command is valid only the first time you use the java asjava.uniproxy.UniProxyAdmin-Client command.
suspend	Suspends an instance of the proxy server. The proxy server stops receiving new connection requests but continues to service all existing connections.
restart	Restarts a stopped instance of the proxy server. Existing connections are unaffected, but the proxy server can receive new connection requests.
shutdown	Shuts down an instance of the proxy server. The shutdown command suspends the proxy server, waits for all connections to disconnect, then terminates the proxy server process.
shutdown:fast	Shuts down an instance of the proxy server. The shutdown command suspends the proxy server, terminates all connections after their current packet exchange finishes (instead of waiting), then terminates the proxy server process.
reconfigure	Reloads the proxy server configuration. The new configuration is effective immediately, but all connection-specific options remain in force for existing connections.
status	Writes the current status of the proxy server to the log file. The status command lists all active connections as well as the current configuration. It also clears the DNS cache.

Proxy Server Commands

Getting Help for the Proxy Server

The following command lists the syntax of the *java* command:

```
java asjava.uniproxy.UniProxyAdminClient help
```

A Tour of the Objects

Code Examples	4-5
Database Account Flavors	4-6
Objects and Methods Quick Reference	4-7
Unijava Methods	4-7
UniSession Methods	4-8
UniFile Methods	4-9
UniDictionary Methods	4-10
UniSequentialFile Object Methods	4-11
UniString Methods	4-12
UniDynArray Methods	4-13
UniRecord Methods	4-13
UniDataSet Methods	4-14
UniStringTokenizer Methods	4-15
UniSelect List Methods	4-15
UniCommand Methods	4-16
UniSubroutine Methods	4-16
UniTransaction Methods	4-17
UniNLSLocale Methods	4-17
UniObjects and BASIC Equivalents	4-18
UniJava Object	4-22
UniJava()	4-22
UniJava Object Methods	4-22
Example Using the UniJava Object	4-23
UniSession Object	4-25
UniSession Object Methods	4-26
Example Using the UniSession Object	4-49
UniFile Object	4-51

UniFile Object Methods	4-51
Example Using the UniFile Object	4-70
UniDictionary Object	4-71
UniDictionary Object Methods	4-71
Example Using the UniDictionary Object	4-96
UniSequentialFile Object	4-97
UniSequentialFile Object Methods	4-97
Example Using the UniSequentialFile Object	4-103
UniString Object	4-106
UniString Object Constructors	4-106
UniString Object Methods	4-107
UniStringTokenizer Object	4-117
UniStringTokenizer Object Constructors	4-117
UniDynArray Object	4-119
UniDynArray Object Constructors	4-119
UniDynArray Object Methods	4-120
Example Using the UniDynArray Object	4-125
UniRecord Object	4-126
UniRecord Object Constructors	4-126
UniRecord Object Methods	4-127
UniDataSet Object	4-129
UniDataSet Object Constructors	4-129
UniDataSet Object Methods	4-129
UniSelectList Object	4-137
UniSelectList Object Methods	4-137
Example Using the UniSelectList Object	4-142
UniCommand Object	4-144
UniCommand Object Methods	4-144
Example Using the UniCommand Object	4-149
UniSubroutine Object	4-150
UniSubroutine Object Methods	4-150
Example Using the UniSubroutine Object	4-153
UniTransaction Object Methods	4-155
Example Using the UniTransaction Object	4-156
UniNLSLocale Object (UniVerse Only)	4-158
UniNLSLocale Object Methods	4-158

Uni-NLSMap Object (UniVerse Only)	4-161
Uni-NLSMap Object Methods	4-161
UniException Object	4-164
UniException Object Methods	4-164
UniXML Object	4-167
Public Methods	4-167

This chapter describes the objects used in UniObjects for Java, together with their associated methods, in the order in which you are most likely to use them in an application.

Object	Description
UniJava	The UniJava object is the starting point for all applications and is used to access all UniSession objects.
UniSession	The UniSession object defines and manages a database session on the server. It controls access to all database objects dependent on it.
UniFile UniDictionary	Next, your program is likely to access a database file on the server through a UniFile or UniDictionary object.
UniSequentialFile	If you want to use data in an operating system file, you use the UniSequentialFile object.
UniString	The UniString object represents UniVerse or UniData data. It includes many string manipulation routines users typically expect in their applications. You can also use this object independently of a session.
UniStringTokenizer	The UniStringTokenizer class acts much like the standard Java StringTokenizer class. It lets you define a token separator character, and automatically parses through the string, returning each individual token. It differs from the standard java class in that it returns an empty record whenever two separator characters appear consecutively.
UniDynArray	You can address records in database files through the UniDynArray object. You can also use this object independently of a session.

Code Examples

The following objects include a short program example that illustrates many of the methods associated with the object: `UniSession`, `UniFile`, `UniDictionary`, `UniSequentialFile`, `UniDynArray`, `UniSelectList`, `UniCommand`, and `UniSubroutine`. **Case-Sensitivity**

The names of the objects and methods used in `UniObjects` for Java are case-sensitive.

Database Account Flavors

On UniVerse systems UniObjects for Java works best in IDEAL flavor UniVerse accounts. In other account flavors, status or error codes returned by some methods may vary from those documented.

On UniData systems ECLTYPE U is best. You may encounter variations with other ECLTYPE or UDT.OPTIONS settings.

Objects and Methods Quick Reference

The following tables are a quick reference to the methods available with each object.

Unijava Methods

The following table lists the Unijava methods.

Primary Methods	Accessor Methods
closeAllSessions()	getMaxSessions()
closeSession()	getNumSessions()
openSession()	getVersionNumber()

Unijava Methods

UniSession Methods

The following table lists the UniSession methods.

Primary Methods	Accessor Methods	Mutator Methods
command()	getAccountPath()	setAccountPath()
connect()	getAtVariable()	setAtVariable()
disconnect()	getCompressionThreshold()	setCompressionThreshold()
dynArray()	getConnectionString()	setConnectionString()
iconv()	getDataSourceType()	setDataSourceType()
nlsLocale()	getDefaultBlockingStrategy()	setDefaultBlockingStrategy()
nlsMap()	getDefaultEncryptionType()	setDefaultEncryptionType()
oconv()	getDefaultLockStrategy()	setDefaultLockStrategy()
open()	getDefaultReleaseStrategy()	setDefaultReleaseStrategy()
openDict()	getDeviceSubkey()	setDeviceSubkey()
openSeq()	getHostName()	setHostName()
releaseTaskLock()	getHostPort()	setHostPort()
selectList()	getHostType()	
setTaskLock()	getMarkCharacter()	
status()	getMaxOpenFiles()	
subroutine()	getNumOpenFiles()	
transaction()	getPassword()	setPassword()
	getProxyHost()	setProxyHost()
	getProxyPort()	setProxyPort()
	getProxyToken()	setProxyToken()
	getTimeout()	setTimeout()
	getTransport()	setTransport()
	getUserName()	setUserName()
	isActive()	
	isCompressionEnabled()	
	isEncryptionEnabled()	
	isNLSEnabled()	
	isNLSLocalesEnabled()	

UniSession Methods

UniFile Methods

The following table lists the UniFile methods.

Primary Methods	Accessor Methods	Mutator Methods
clearFile()	getBlockingStrategy()	setBlockingStrategy()
close()	getEncryptionType()	setEncryptionType()
deleteRecord()	getFileName()	setFileName()
getAkInfo()	getFileType()	
iType()	getLockStrategy()	setLockStrategy()
lockFile()	getRecord()	setRecord()
lockRecord()	getRecordID()	setRecordID()
open()	getReleaseStrategy()	setReleaseStrategy()
read()	isOpen()	
readField()	isRecordLocked()	
readNamedField()		
status()		
unlockFile()		
unlockRecord()		
write()		
writeField()		
writeNamedField()		

UniFile Methods

UniDictionary Methods

The following table lists the UniDictionary methods.

Primary Methods	Accessor Methods	Mutator Methods
clearFile()	getAssoc()	setAssoc()
close()	getBlockingStrategy()	setBlockingStrategy()
deleteRecord()	getConv()	setConv()
getAkInfo()	getEncryptionType()	setEncryptionType()
iType()	getFileName()	setFileName()
lockFile()	getFileType()	
lockRecord()	getFormat()	setFormat()
open()	getLoc()	setLoc()
read()	getLockStrategy()	setLockStrategy()
readField()	getName()	setName()
readNamedField()	getRecord()	setRecord()
status()	getRecordID()	setRecordID()
unlockFile()	getReleaseStrategy()	setReleaseStrategy()
unlockRecord()	getSM()	setSM()
write()	getSQLType()	setSQLType()
writeField()	getType()	setType()
writeNamedField()	isOpen()	

UniDictionary Methods

UniSequentialFile Object Methods

The following table lists the UniSequential Object methods.

Primary Methods	Accessor Methods	Mutator Methods
close()	getEncryptionType()	setEncryptionType()
fileSeek()	getReadSize()	setReadSize()
open()	getTimeout()	setTimeout()
readBlk()	isOpen()	
readLine()		
status()		
writeBlk()		
writeEOF()		
writeLine()		

UniString Methods

The following table lists the UniString methods.

Primary Methods	Accessor Methods	Mutator Methods
alpha()	getBytes()	
append()	getChars()	
change()	getMarkCharacter()	
charAt()		setCharAt()
compareTo()		setValue()
convert()		
count()		
dcount()		
equals()		
equalsIgnoreCase()		
iconv()		
insert()		
left()		
oconv()		
quote()		
right()		
status()		
substring()		
toCharArray()		
toLowerCase()		
toString()		
toUpperCase()		

UniString Methods

UniDynArray Methods

The following table describes the UniDynArray methods.

Primary Methods
count()
dcount()
delete()
extract()
insert()
length()
remove()
replace()
toString()

UniDynArray Methods

UniRecord Methods

The following table lists the UniRecord methods.

Primary Methods	Accessor Methods	Mutator Methods
returnCode()	getRecord()	setRecord()
status()	getRecordID()	setRecordID()
toString()		setReturnCode()
toUniString()		setStatus()
toUniDynArray()		

UniRecord Methods

UniDataSet Methods

The following table lists the UniDataSet methods.

Primary Methods	Accessor Methods
absolute()	getCurrentRow()
afterLast()	getDataSet()
append()	getIDSet()
close()	getRowCount()
deleteRow()	isAfterLast()
findRow()	isBeforeFirst()
first()	isFirst()
toString()	isLast()
toUniDynArray()	
toUniString()	
toUniRecord()	
insert()	
last()	
next()	
previous()	
relative()	
setIndex()	
toString()	

UniDataSet Methods

UniStringTokenizer Methods

The following table lists the UniString Tokenizer methods.

Primary Methods

countTokens()
hasMoreTokens()
nextToken()
resetTokenizer()

UniString Tokenizer Methods

UniSelect List Methods

The following table lists the UniSelect List methods.

Primary Methods	Accessor Methods	Mutator Methods
clearList()	getEncryptionType()	setEncryptionType()
formList()	getList()	
next()	isLastRecordRead()	
readList()		
saveList()		
select()		
selectAlternateKey()		
selectMatchingAK()		

UniSelect List Methods

UniCommand Methods

The following table lists the UniCommand methods.

Primary Methods	Accessor Methods	Mutator Methods
cancel()	getAtSelected()	
exec()	getBlockSize()	setBlockSize()
nextBlock()	getCommand()	setCommand()
reply()	getEncryptionType()	setEncryptionType()
response()	getSystemReturnCode()	
	status()	

UniCommand Methods

UniSubroutine Methods

The following table lists the UniSubroutine methods.

Primary Methods	Accessor Methods	Mutator Methods
call()	getArg()	setArg()
resetArgs()	getEncryptionType()	setEncryptionType()
	getNumArgs()	setNumArgs()
	getRoutineName()	setRoutineName()
	status()	

UniSubroutine Methods

UniTransaction Methods

The following table lists the UniTransaction methods.

Primary Methods	Accessor Methods
commit()	getLevel()
rollback()	isActive()
begin()	

Unitransaction Methods

UniNLSLocale Methods

The following table describes the UniNLSLocate methods.

Primary Methods	Accessor Methods
setName()	getClientNames() getServerNames()

UniNLSLocale Methods

UniObjects and BASIC Equivalents

The following table shows the UniObjects for Java methods and their equivalents in UniObjects and BASIC.

Method	UniObjects Equivalent	BASIC Equivalent
call()	Call	CALL
cancel()	Cancel	No direct equivalent
clearFile()	ClearFile	CLEARFILE
clearList()	ClearList	CLEARSELECT
close()	CloseFile	CLOSE, reassignment to file variable CLOSESEQ
	CloseSeqFile	
commit()	Commit	COMMIT (UniVerse) TRANSACTION COMMIT (UniData)
connect()	Connect	No direct equivalent
count()	Count	COUNT()
dcount()	dcount	DCOUNT()
del()	Del	DEL
deleteRecord()	DeleteRecord	DELETE, DELETEU
disconnect()	Disconnect	No direct equivalent
exec()	Exec	EXECUTE
field()	Field	No direct equivalent
fileSeek()	FileSeek	SEEK
formList()	FormList	FORMLIST

UniObjects for Java Methods and Their Equivalents

Method	UniObjects Equivalent	BASIC Equivalent
getAkInfo()	GetAkInfo	INDICES()
getArg()	GetArg	No direct equivalent
getAtVariable()	GetAtVariable	No direct equivalent
getList()	GetList	No direct equivalent
iconv()	Iconv	ICONV()
ins()	Ins	INS
isActive()	IsActive	No direct equivalent
isOpen()	IsOpen	No direct equivalent
iType()	IType	ITYPE()
length()	Length	No direct equivalent
lockFile()	LockFile	FILELOCK
lockRecord()	LockRecord	RECORDLOCKL RECORDLOCKU
next()	Next	READNEXT
nextBlock()	NextBlock	No direct equivalent
oconv()	Oconv	OCONV()
open()	OpenFile OpenDictionary OpenSequential	OPEN OPEN DICT OPENSEQ
read()	Read	READ, READL, READU
readBlk()	ReadBlk	READBLK
readField()	ReadField	READV, READVL, READVU
readLine()	ReadLine	READSEQ

UniObjects for Java Methods and Their Equivalents (Continued)

Method	UniObjects Equivalent	BASIC Equivalent
readList()	ReadList	READLIST
readNamedField() ()	ReadNamedField	No direct equivalent.
releaseTaskLock() k()	ReleaseTaskLock	UNLOCK
replace()	Replace	REPLACE()
reply()	Reply	No direct equivalent
resetArgs()	ResetArgs	No direct equivalent
rollback()	Rollback	ROLLBACK (UniVerse) TRANSACTION ABORT (UniData)
saveList()	SaveList	No direct equivalent
select()	Select	SELECT
selectAlternateKey() eKey()	SelectAlternateKey	SELECTINDEX
selectList()	SelectList	No direct equivalent
selectMatchingAk() Ak()	SelectMatchingAk	SELECTINDEX
setArg()	SetArg	No direct equivalent
setAtVariable()	SetAtVariable	No direct equivalent
setName()	SetName	No direct equivalent
setTaskLock()	SetTaskLock	LOCK
start()	Start	BEGIN TRANSACTION (UniVerse) TRANSACTION START (UniData)
subroutine()	Subroutine	No direct equivalent
subValue()	SubValue	No direct equivalent

UniObjects for Java Methods and Their Equivalents (Continued)

Method	UniObjects Equivalent	BASIC Equivalent
unlockFile()	UnlockFile	FILEUNLOCK
unlockRecord()	UnlockRecord	RELEASE
value()	Value	No direct equivalent
write()	Write	WRITE, WRITEU
writeBlk()	WriteBlk	WRITEBLK
writeEOF()	WriteEOF	WEOFSEQ
writeField()	WriteField	WRITEV, WRITEVU
writeLine()	WriteLine	WRITESEQ
writeNamedField() d()	WriteNamedField	No direct equivalent

UniObjects for Java Methods and Their Equivalents (Continued)

UniJava Object

The UniJava object manages and enumerates all UniSession objects. It does this by creating a Vector, `uniSessionsVector`, which holds all UniSession object references. When a new UniSession object is created (through the `openSession()` method), it is added to `uniSessionsVector`. When a session is closed, the UniSession object is removed from `uniSessionsVector`.

Each application must create one UniJava object.

UniJava()

This is the default constructor for the class. It initializes the `uniSessionsVector`.

UniJava Object Methods

These are the methods you can use with the UniJava object:

- `closeAllSessions()`
- `closeSession()`
- `getMaxSessions()`
- `getNumSessions()`
- `getVersionNumber()`
- `openSession()`

public void closeAllSessions() throws UniSessionException

This method closes all open sessions by calling the `UniSession.disconnect()` method for each session in `uniSessionsVector` and removing them.

public void closeSession (UniSession aSession) throws UniSessionException

This method closes the specified session by calling the `UniSession.disconnect()` method and removing the session entry from `uniSessionsVector`.

`UniSession aSession` is the `UniSession` object you want to close.

public int getMaxSessions()

This method returns the maximum number of sessions that can be open at one time. If it returns 0, there is no maximum.

public int getNumSessions()

This method returns the number of currently open sessions.

public String getVersionNumber()

This method returns the current version number of UniObjects for Java as a `String`.

public UniSession openSession() throws UniSessionException

This method opens a new `UniSession` object. If an error occurs when trying to open a session, or if the number of open sessions exceeds the maximum number of sessions allowed, `openSession()` throws `UniSessionException`.

Example Using the UniJava Object

```
import asjava.uniobjects.*;
public class UOJTest
{
    public static void main( String args[])
    {
        UniJava uJava = new UniJava();
        System.out.println("Version number = " +
        uJava.getVersionNumber() );
        try
        {
            UniSession uSession = uJava.openSession();
        }
        catch ( UniSessionException e )
        {
            System.out.println( e.getExtendedText() );
        }
    }
}
```

UniSession Object

The `UniSession` object defines and manages a database session on the server. It is the central object for any database session, controlling access to all objects dependent on it. You can define multiple sessions up to the number specified by the `UniJava.getMaxSessions()` method.

UniSession()

This is the default constructor for this class. You define a `UniSession` object through the `UniJava.openSession()` method. You access other objects on the server, such as `UniFile` or `UniCommand` objects through the `UniSession` object. See [Opening a Database Session](#) in Chapter 2, “[Using UniObjects for Java](#),” for information about using the `UniSession` object. For a program example, see [“Example Using the UniSession Object”](#) on page 4-48.

UniSession Object Methods

These are the methods you can use with the UniSession object:

- `command()`
- `connect()`
- `disconnect()`
- `dynArray()`
- `getAccountPath()`
- `getAtVariable()`
- `getCompressionThreshold()`
- `getConnectionString()`
- `getDataSource()`
- `getDefaultBlockingStrategy()`
- `getDefaultEncryptionType()`
- `getDefaultLockStrategy()`
- `getDefaultReleaseStrategy()`
- `getDeviceSubkey()`
- `getHostName()`
- `getHostPort()`
- `getHostType()`
- `nlsLocale()`
- `nlsMap()`
- `oconv()`
- `open()`
- `openDict()`
- `openSeq()`
- `releaseTaskLock()`
- `selectList()`
- `setAccountPath()`
- `setAtVariable()`
- `setCompressionThreshold()`
- `setConnectionString()`
- `setDataSourceType()`
- `setDefaultBlockingStrategy()`
- `setDefaultEncryptionType()`
- `getMarkCharacter()`
- `getMaxOpenFiles()`
- `getNumOpenFiles()`
- `getPassword()`
- `getProxyHost()`
- `getProxyPort()`
- `getProxyToken()`
- `getServerVersion()`
- `getTimeout()`
- `getTransport()`
- `getUserName()`
- `iconv()`
- `isActive()`
- `isCompressionEnabled()`
- `isEncryptionEnabled()`
- `isNLSEnabled()`
- `isNLSLocaleEnabled()`
- `setDefaultLockStrategy()`
- `setDefaultReleaseStrategy()`
- `setDeviceSubkey()`
- `setHostName()`
- `setHostPort()`
- `setPassword()`
- `setProxyHost()`
- `setProxyPort()`
- `setProxyToken()`
- `setTaskLock()`
- `setTimeout()`
- `setTransport()`
- `setUserName()`
- `status()`
- `subroutine()`
- `transaction()`

UniSession Object Methods

public UniCommand command () throws UniSessionException

public UniCommand command (Object aCommandString) throws UniSession Exception

This method creates a single UniCommand object for the session.

Object aCommandString is the command to run.

public void connect () throws UniSessionException

public void connect (Object aHost, Object aUserName, Object aPassword, Object aPath) throws UniSessionException

public void connect (Object aHost, int aHostPort, Object aUserName, Object aPassword, Object aPath) throws UniSessionException

public void connect (Object aHost, Object aUserName, Object aPassword, Object aPath, Object aProxyHost, Object aProxyPort, Object aProxyToken) throws UniSessionException

public void connect (Object aHost, int aHostPort, Object aUserName, Object aPassword, Object aPath, Object aProxyHost, Object aProxyToken) throws UniSessionException

public void connect (Object aHost, Object aUserName, Object aPassword, Object aPath, Object aProxyHost, Object aProxyToken) throws UniSessionException

This method opens a session on the server.

Object aHost is the name of the host to connect to, corresponding to the host name accessible through the getHostName () and setHostName () methods.

`int aHostPort` is the port number of the host to connect to, corresponding to the host port number accessible through the `getHostPort ()` and `setHostPort ()` methods.

Object `aUserName` is the user name on the server, corresponding to the user name accessible through the `getUserName ()` and `setUserName ()` methods.

Object `aPassword` is the password on the server, corresponding to the password accessible through the `getPassword ()` and `setPassword ()` methods.

Object `aPath` is the path of the account on the server, corresponding to the account path accessible through the `getAccountPath ()` and `setAccountPath ()` methods.

Object `aProxyHost` is the name of the proxy host to connect to, corresponding to the proxy host accessible through the `getProxyHost ()` and `setProxyHost ()` methods.

`int aProxyPort` is the number of the proxy port to connect to, corresponding to the proxy port number accessible through the `getProxyPort ()` and `setProxyPort ()` methods.

Object `aProxyToken` is the security token to use when connecting to a proxy host. You can also use the `setProxyToken ()` method to set this token.

If you do not specify any parameters, values are those set up by the `setHostName ()`, `setHostPort ()`, `setUserName ()`, `setPassword ()`, `setAccountPath ()`, `setProxyHost ()`, `setProxyPort ()`, and `setProxyToken ()` methods.

If the client cannot connect to the server, `connect ()` throws `UniSessionException`.

This method corresponds to the UniObjects **Connect** method.

The following examples show two ways of connecting to a server:

```
UniSession uSession = uJava.openSession();
uSession.setHostName ("alfa");
uSession.setUserName ("davem");
uSession.setPassword ("password");
uSession.setAccountPath ("/usr/uv/sales");
uSession.connect ();
UniSession uSession = uJava.openSession();
uSession.connect ("alfa", "davem", "password", "/usr/uv/sales");
```



public void disconnect () throws UniSessionException

This method closes an active session, closes any open files, and releases any locks associated with the session. After calling this method, the `isActive()` method returns `false`, and any operation performed on this session, other than `connect()` or `disconnect()`, results in an error and throws `UniSessionException`.

Note: Other objects created by or associated with the session are still available, but using them may cause an error. For example, if you have a `UniFile` object created by the `UniSession` object, you can access the last record that was read from the file, but you cannot read another record.

This method corresponds to the `UniObjects` **Disconnect** method.

The following example disconnects the current session from the server:

```
try
{
    client.disconnect();
    client = null;
}
catch (UniSessionException e)
{
    printStr(e.getMessage() + "=" + e.getErrorCode() );
    return;
}
```

public UniDynArray dynArray()

public UniDynArray dynArray (Object aString)

This method returns a new `UniDynArray` object either as a default string or as the string specified by `aString`.

Object `aString` is the string you want to create as a dynamic array.

The returned `UniDynArray` object inherits the system delimiters associated with this session.

public String getAccountPath()

This method returns the account path, which is the name of the database account to which the session is connected. This method corresponds to the `UniObjects` **AccountPath** property.

***public String getAtVariable (int aTokenVal) throws
UniSessionException***

This method returns the value of a BASIC @variable as a string.

int aTokenVal is the @variable whose value is to be retrieved, which is one of the following.

Value	Token	BASIC @Variables
1	UniObjectsTokens.AT_LOGNAME	@LOGNAME
2	UniObjectsTokens.AT_PATH	@PATH
3	UniObjectsTokens.AT_USERNO	@USERNO
4	UniObjectsTokens.AT_WHO	@WHO
5	UniObjectsTokens.AT_TRANSACTION	@TRANSACTION
6	UniObjectsTokens.AT_DATA_PENDING	@DATA.PENDING
7	UniObjectsTokens.AT_USER_RETURN_CODE	@USER.RETURN.CODE
8	UniObjectsTokens.AT_SYSTEM_RETURN_CODE	@SYSTEM.RETURN.CODE
9	UniObjectsTokens.AT_NULL_STR	@NULL.STR
10	UniObjectsTokens.AT_SCHEMA	@SCHEMA

int aTokenVal @variables

This method corresponds to the UniObjects **GetAtVariable** method.

public int getCompressionThreshold()

This method returns the current compression threshold in bytes. The compression threshold is set only if isCompressionEnabled() returns true.

Data compression is done on packets whose data size exceeds the compression threshold. 0 is the default, which means no compression is done.

public String getConnectionString()

This method returns the connection string used for the connection. This connection string is DEFCS by default and corresponds to the entry used in the *unirpcservices* file on the server to launch the backend server process.

public String getDataSourceType()

This method returns the data source type connected to, currently either: UNIVERSE or UNIDATA.

public int getDefaultBlockingStrategy()

This method returns the current default blocking strategy value, which is one of the following.

Value	Token	Description
1	UniObjectsTokens.UVT_WAIT_LOCKED	If the record is locked, wait until it is released.
2	UniObjectsTokens.UVT_RETURN_LOCKED	Return a status value indicating the state of the lock. This is the default value.

getDefaultBlockingStrategy Return Values

This method corresponds to the UniObjects **DefaultBlockingStrategy** property.

public int getDefaultEncryptionType()

This method returns the current default encryption value, which is one of the following.

Value	Token	Description
0	UniObjectsTokens.NO_ENCRYPT	No encryption. This is the default value.
1	UniObjectsTokens.UV_ENCRYPT	Encrypt all data using internal database encryption.

getDefaultEncryptionType Return Values

public int getDefaultLockStrategy()

This method returns the current default locking strategy value, which is one of the following:

Value	Token	Description
0	UniObjectsTokens.NO_LOCKS	No locking. This is the default value.
1	UniObjectsTokens.UVT_EXCLUSIVE_READ	Sets an exclusive update lock (READU).
2	UniObjectsTokens.UVT_SHARED_READ	Sets a shared read lock (READL).

getDefaultLockStrategy Return Values

This method corresponds to the UniObjects **DefaultLockStrategy** property.

public int getDefaultReleaseStrategy()

This method returns the current default release strategy value, which is one of the following.

Value	Token	Description
1	UniObjectsTokens.WRITE_RELEASE	Releases the lock when the record is written. This is the property's initial value.

getDefaultReleaseStrategy Return Values

Value	Token	Description
2	UniObjectsTokens.UVT_READ_RELEASE	Releases the lock when the record is read.
4	UniObjectsTokens.UVT_EXPLICIT_RELEASE	Maintains locks as specified by the <code>setLockStrategy()</code> method. You can release locks only with the <code>unlockRecord()</code> method.
8	UniObjectsTokens.UVT_CHANGE_RELEASE	Releases the lock whenever a new value is set by the <code>setRecordId()</code> method.

getDefaultReleaseStrategy Return Values (Continued)

This method corresponds to the UniObjects **DefaultReleaseStrategy** property.

public String getDeviceSubkey()

This method gets the device subkey being used. The subkey is used for device licensing.

public String getHostName()

This method returns the name of the database server as specified either by the `setHostName()` method or by direct reference in the `connect()` method.

This method corresponds to the UniObjects **HostName** property.

public int getHostPort()

This method returns the value of the port number on the host to use for the connection. You can set the port number using the `setHostPort()` method or by referencing it directly in the `connect()` method.

public int getHostType()

This method returns the current type of host the session is connected to, which is one of the following:

Value	Token	Description
0	UniObjectsTokens.UVT_NONE	The host system cannot be determined, the session is not connected.
1	UniObjectsTokens.UVT_UNIX	The host is a UNIX system.
2	UniObjectsTokens.UVT_NT	The host is a Windows system.

getHostType Return Values

This method corresponds to the UniObjects **HostType** property.

public String getMarkCharacter (int aTokenVal) throws UniSessionException

This method returns the value of the specified system delimiter. Call this method, especially with an NLS-enabled server, to determine what are the proper values for each system delimiter.

`int aTokenVal` is the system delimiter you want. It is one of the following.

Value	Token	Description
1	UniObjectsTokens.IM	Item mark
2	UniObjectsTokens.FM	Field mark
3	UniObjectsTokens.VM	Value mark
4	UniObjectsTokens.SM	Subvalue mark
5	UniObjectsTokens.TM	Text mark
6	UniObjectsTokens.SQLNULL	Null value

getMarkCharacter Values

This method corresponds to the UniObjects **FM**, **IM**, **SQLNULL**, **SVM**, **TM**, and **VM** properties.

public int getMaxOpenFiles()

This method returns the maximum number of files that can be open at any one time.

public int getNumOpenFiles()

This method returns the number of files that are currently open.

public String getPassword()

This method returns the current password as set by the `setPassword()` method. It corresponds to the UniObjects **Password** property.

public String getProxyHost()

This method returns the proxy host name as specified by the `setProxyHost()` method.

public int getProxyPort()

This method returns the port number used to connect to the proxy host. You can set the proxy port number using the `setProxyPort()` method or by referencing it directly in the `connect()` method.

public String getProxyToken()

This method returns the value set by the security token for proxy connections.

public int getServerVersion()

This method returns the version of the backend server. A value of less than 2 represents a pre-Release 9.5 server.

public int getTimeout()

This method returns the current RPC timeout value as set by the `setTimeout()` method. It corresponds to the UniObjects **Timeout** property.

public int getTransport()

This method returns the current transport type. Currently only TCP/IP connections are allowed.

This method corresponds to the UniObjects **Transport** property.

public String getUsername()

This method returns the current user name as set by the `setUsername()` method. It corresponds to the UniObjects **UserName** property.

public UniString iconv (Object aStringVal, Object convCode) throws UniStringException

This method converts an input string to internal storage format.

Object `aStringVal` is the string to convert.

Object `convCode` is any BASIC ICONV conversion code.

The `iconv()` method sets the UniSession object's status to one of the following values:

Value	Description
0	The conversion was successful.
1	The string supplied was invalid.
2	The conversion code supplied was invalid.
3	Conversion of possibly invalid data was successful.

UniSession Object Status

This method corresponds to the UniObjects **Iconv** method and the BASIC ICONV function.

```
try{
    UniString iDate = uSession.iconv( "12 Oct 96", "D2/" );
} catch (UniStringException e )
{ ... deal with error...
}
```

public boolean isActive()

This method returns `true` if a connection was successfully established. It returns `false` if an error occurred. It corresponds to the UniObjects **IsActive** method.

public boolean isCompressionEnabled()

This method returns `true` if this connection supports data compression, `false` otherwise.

Note: Compression is not available when connecting to servers running a release of UniVerse earlier than Release 9.5.

public boolean isEncryptionEnabled()

This method returns `true` if encryption is enabled for this connection, `false` otherwise.

Note: Encryption is not available when connecting to servers running a release of UniVerse earlier than Release 9.5.

public boolean isNLSEnabled()

This method returns `true` if a connection is NLS-ready. It returns `false` if the current connection does not support NLS.

public boolean isNLSLocalesEnabled()

This method returns `true` if both NLS and NLS locales are enabled on the server. It returns `false` if they are not enabled.

public UniNLSLocale nlsLocale () throws UniSessionException

This method returns a `UniNLSLocale` object. Use the `isNLSEnabled()` method to determine if NLS is enabled.

This method corresponds to the UniObjects **NLSLocale** property.



public Uni-NLSMap nlsMap () throws UniSessionException

This method returns a Uni-NLSMap object. Use the isNLSEnabled() method to determine if NLS is enabled.

This method corresponds to the UniObjects **NLSMap** property.

public UniString oconv (Object aStringVal, Object convCode) throws UniStringException

This method converts a string from the internal storage format to an external format defined by a conversion code.

Object aStringVal is the string to convert.

Object convCode is any BASIC OCONV conversion code.

The oconv () method returns one of the following status values:

Value	Description
0	The conversion was successful.
1	The string supplied was invalid.
2	The conversion code supplied was invalid.
3	Successful conversion of possibly invalid data.

oconv Method Return Values

This method corresponds to the UniObjects **Oconv** method and the BASIC OCONV function.

```
try{
  UniString oDate = uSession.oconv( iDate, "D2/" );
} catch ( UniStringException e )
{ ... deal with exception
}
```

public UniFile open (Object aFileName) throws UniSessionException

This method opens an existing database file and returns a UniFile object, allowing access to the file.

Object `aFileName` is the name of the database file to open.

This example opens the ORDERS file:

```
UniFile uFile = uSession.open ("ORDERS");
```

This method corresponds to the UniObjects **OpenFile** method and the BASIC OPEN statement.

public UniDictionary openDict (Object aFileName) throws UniSessionException

This method opens an existing database file and returns a `UniFile` object, allowing access to the file.

Object `aFileName` is the name of the file dictionary to open.

This example opens the dictionary of the ORDERS file:

```
UniDictionary uDict = uSession.openDict("ORDERS");
```

This method corresponds to the UniObjects **OpenDictionary** method and the BASIC OPEN statement.

public UniSequentialFile openSeq (Object aFileName, Object aRecordID, boolean aCreateFlag) throws UniSessionException

Note: UniObjects for Java cannot process UniData files sequentially. The following method applies only to UniVerse databases.

This method opens a file for sequential processing and returns a `UniSequentialFile` object.

Object `aFileName` is the name of an existing type 1 or type 19 file.

Object `aRecordID` is a record in the file. If the record does not exist and if `aCreateFlag` is true, `open` creates it.

boolean `aCreateFlag` is a flag specifying that the record should or should not be created if it does not exist. If `aCreateFlag` is true, `open()` creates the record.



If the record cannot be opened owing to an error on the server, `openSeq()` throws `UniSessionException`, and the `UniSession` object's status is one of the following values:

Value	Description
0	No record ID was found.
1	The specified file is not type 1 or type 19.
2	The specified file was not found.

UniSessionobject Status

This example opens the TEST2 program file for sequential processing:

```
UniSequentialFile uSeq = uSession.openSeq("BP", "TEST2", false);
```

This method corresponds to the database CREATE command (if the create flag is set to `true`), the UniObjects **OpenSequential** method, and the BASIC OPENSEQ statement.

public void releaseTaskLock (int aLockNumber) throws UniSessionException

This method releases one of the 64 UniVerse task locks. For more information about task locks, see [Task Locks](#) in Chapter 2, “Using UniObjects for Java.”

`int aLockNumber` is the number, 0 through 63, of the task lock to release.

This method corresponds to the UniObjects **ReleaseTaskLock** method.

```
uSession.releaseTaskLock( 4 );
```

public UniSelectList selectList (int aSelectListNumber) throws UniSessionException

This method returns a `UniSelectList` object representing one of the 11 UniVerse select lists.

`int aSelectListNumber` is the number, 0 through 10, of the select list to use.

This example creates active select list 0:

```
UniSelectList uSelect = uSession.selectList(0);
```

This method corresponds to the UniObjects **SelectList** method.

public void setAccountPath (Object aAccountPath)

This method sets the name of the account to connect to on the database server.

Object `aAccountPath` is the path of the account, which you can specify as:

- A full path. For example, on a Windows server:

`d:/uv/sales/customer`

Or on a UNIX server:

`/usr/uv/sales/CUSTOMER`

- A valid account name as specified in the UV.ACCOUNT file on the server.

You can set this property at design time or run time. If you do not set it before calling the `connect ()` method, you are prompted for this value in a dialog box.

This method corresponds to the UniObjects **AccountPath** property.

public void setAtVariable (int aTokenVal, Object aAtVariable) throws UniSessionException

This method sets the value of a BASIC @variable to a string.

int `aTokenVal` is the @variable name to be set:

Value	Token	Description
7	UniObjectsTokens.AT_USER_RETURN_CODE	@USER.RETURN.CODE

`aTokenVal @variable`

Object `aAtVariable` is the string value to which the @variable is to be set.

This method corresponds to the UniObjects **SetAtVariable** method.

public void setCompressionThreshold (int aCompressionThresholdVal) throws UniSessionException

This method sets the threshold for data compression.

`int aCompressionThresholdVal` is the compression threshold in bytes. 0 is the default, which means no compression is done.

Data compression is done on packets whose data size exceeds the value of `aCompressThresholdVal`. Compression and decompression of data is handled at the network comms level and is not accessible to users.

public void setConnectionString (String connString)

This method sets the connection string to be used for connections to remote server. This string corresponds to an entry in the *unirpcservices* file on the server, which tells the system which process to launch when requested. By default, this value is set to DEFCS.

public void setDataSourceType (String dsType) throws UniSessionException

This method sets the type of system being connected to. This can be only UNIVERSE or UNIDATA. It affects the connection string being used.

public void setDefaultBlockingStrategy (int aBlockingStrategy) throws UniSessionException

This method sets the default blocking strategy for all `UniFile` and `UniDictionary` objects.

`int aBlockingStrategy` is one of the following:

Value	Token	Description
1	<code>UniObjectsTokens.UVT_WAIT_LOCKED</code>	If the record is locked, wait until it is released (see Note).
2	<code>UniObjectsTokens.UVT_RETURN_LOCKED</code>	Return a status value indicating the state of the lock. This is the default value.

aBlockingStrategy Values



Changing the blocking strategy does not affect existing UniFile or UniDictionary objects.

Note: Use the `UniObjectsTokens.UVT_WAIT_LOCKED` value with caution. While the method is waiting for the lock to be released, your client window is effectively frozen and will not respond to mouse clicks.

This method corresponds to the UniObjects **DefaultBlockingStrategy** property.

public void setDefaultEncryptionType (int aEncryptType) throws UniSessionException

This method sets the default encryption to use for the session.

`int aEncryptType` is the type of encryption to use, which is one of the following:

Value	Token	Description
0	<code>UniObjectsTokens.NO_ENCRYPT</code>	No encryption. This is the default value.
1	<code>UniObjectsTokens.UV_ENCRYPT</code>	Encrypt all data using internal database encryption.

aEncryptType Values

If you set `UV_ENCRYPT` for a session, all data transferred between client and server is encrypted.

public void setDefaultLockStrategy (int aLockingStrategy) throws UniSessionException

This method sets the default locking strategy for all UniFile and UniDictionary objects.

`int aLockingStrategy` is one of the following:

Value	Token	Description
0	<code>UniObjectsTokens.NO_LOCKS</code>	No locking. This is the default value.
1	<code>UniObjectsTokens.UVT_EXCLUSIVE_READ</code>	Sets an exclusive update lock (READU).
2	<code>UniObjectsTokens.UVT_SHARED_READ</code>	Sets a shared read lock (READL).

aLockingStrategy Values

Altering the lock strategy does not affect files or dictionaries that are already open.

This method corresponds to the `UniObjects` **DefaultLockStrategy** property.

public setDefaultReleaseStrategy (int aReleaseStrategy) throws UniSessionException

This method sets the default release strategy for all `UniFile` and `UniDictionary` objects. Whenever the record ID is reset with the `setRecordID()` method, the release strategy reverts to the initial value. Altering the release strategy does not affect files or dictionaries that are already opened.

`int aReleaseStrategy` is one of the following:

Value	Token	Description
1	<code>UniObjectsTokens.WRITE_RELEASE</code>	Releases the lock when the record is written. This is the default value.
2	<code>UniObjectsTokens.UVT_READ_RELEASE</code>	Releases the lock when the record is read.
4	<code>UniObjectsTokens.UVT_EXPLICIT_RELEASE</code>	Maintains locks as specified by the <code>setLockStrategy()</code> method. Locks can be released only with the <code>unlockRecord()</code> method.
8	<code>UniObjectsTokens.UVT_CHANGE_RELEASE</code>	Releases the lock whenever a new value is set by the <code>setRecordID()</code> method.

aReleaseStrategy Values

All the values are additive. If you specify `EXPLICIT_RELEASE` with `WRITE_RELEASE` and `READ_RELEASE`, it takes a lower priority. The initial release strategy value is 12, that is, release locks when the record ID changes or when locks are released explicitly.

This method corresponds to the `UniObjects` **ReleaseStrategy** property.

public void setDeviceSubkey (String subKey)

This method sets up the device subkey to use during the initial connection. It is used for device licensing.

public void setHostName (Object aHostName)

This method sets the name of the database server.

Object `aHostName` is the name of the host to connect to.

You can set the host name at design time or run time. If you do not set the value before calling `connect ()`, the `connect ()` method prompts the user for the value so that it can establish a connection to the server.

If you used the `setTransport ()` method to specify a TCP/IP connection, you can use the `setHostName ()` method to specify the IP address, port number, or both, to use for the connection. For example:

- If you enter *server name* (for example, `server1`), a TCP/IP connection is made to that node name.
- If you enter *IP address* (for example, `192.34.56.94`), a TCP/IP connection is made to the specified address.

This method corresponds to the UniObjects **HostName** property.

public void setHostPort (int aHostPort)

This method sets the port number to connect to on the host.

`int aHostPort` is the port number on the host to connect to.

public void setPassword (Object aPassword)

This method sets a password (if required) for the user specified in the `UserName` property.

`Object aPassword` is the password to use.

On Windows servers, this property is ignored, but it should be specified as an empty string if you want your code to be portable to both Windows and UNIX systems.

This method corresponds to the UniObjects **Password** property.

public void setProxyHost (Object aHostName)

This method sets the name of the proxy host to connect to. You must set the proxy host name before you call the `connect ()` method.

`Object aHostName` is the name of the proxy host.

- If you enter *proxy name* (for example, `server1`), a TCP/IP connection is made to that node name.

- If you enter *IP address* (for example, 192.34.56.94), a TCP/IP connection is made to the specified address.

A proxy host functions as an intermediate server that routes packets. Use a proxy host when you want to pass data through a firewall or, in the case of an applet, to route data to a server other than the one serving the applet.

public void setProxyPort (int aProxyPort)

This method sets the port number to connect to on the proxy server.

`int aProxyPort` is the port number on the proxy server to connect to.

public void setProxyToken (String aToken)

This method sets the security token used by a proxy connection.

`String aToken` is the proxy security token.

public void setTaskLock (int aLockNumber) throws UniSessionException

This method locks one of the database's 64 task locks. For more information about task locks, see [Task Locks](#) in Chapter 2, "Using UniObjects for Java."

`int aLockNumber` is the number, 0 through 63, of the task lock to be set.

This method corresponds to the UniObjects **SetTaskLock** method and the BASIC LOCK statement.

```
uSession.setTaskLock( 4 );
```

public void setTimeout (int aTimeoutVal) throws UniSessionException

This method specifies the length of the timeout for a session. The remote procedure call utility (UniRPC) uses the timeout period.

`int aTimeoutVal` is the timeout value in seconds. The default value is 300 seconds (5 minutes).



Note: If you enter a value that is too small, a running process (for example, the `read()` method) may time out. If this occurs, an error code is returned and the connection to the server is dropped.

This method corresponds to the UniObjects **Timeout** property.

public void setTransport (int aTransportType)

This method specifies the transport type to use when connecting to a server.

int aTransportType is:

Value	Token	Description
0	UniObjectsTokens.NETWORK_DEFAULT	TCP/IP

aTransport Type Values

Note: With TCP/IP connections you must enter security information to connect to the server, for example, user name and password.

This method corresponds to the UniObjects **Transport** property.

public void setUsername (Object aUserName)

This method sets the user name to use to log on to a database server.

Object aUserName is a user's login name.

On Windows servers this property is ignored, but you should specify it as an empty string if you want your code to be portable to both Windows and UNIX systems.

This method corresponds to the UniObjects **UserName** property.

public int status()

This method returns the current status value. Refer to each method for a description of the status values that are returned.

This method corresponds to the UniObjects **Status** property.



public UniSubroutine subroutine (Object aSubroutineName, int aNumOfArgs) throws UniSessionException

This method returns a `UniSubroutine` object that calls a cataloged subroutine on the server.

Object `aSubroutineName` is the name of the subroutine, which is the name used when the subroutine was cataloged on the server.

int `aNumOfArgs` is the number of arguments that the server subroutine uses.

This example calls the subroutine `READ.CONFIG`:

```
UniSubroutine uSub = uSession.subroutine("READ.CONFIG", 3);
```

This method corresponds to the `UniObjects Subroutine` method.

public UniTransaction transaction () throws UniSessionException

This method returns a `UniTransaction` object for this session. It corresponds to the `UniObjects Transaction` property.

Example Using the UniSession Object

```
import asjava.uniobjects.*;
public class UOJtest{
    public static void main( String args[])
    {
        UniJava uJava = new UniJava();
        UniSession uSession = uJava.openSession();
        uSession.connect( "alfa", "davem", "password", "/usr/uv/sales"
    );

        UniCommand runCmd = uSession.command( "RUN BP FOO" );

        /* do actual work */
        uSession.disconnect();
    } catch ( UniSessionException e )
    { /* handle exception */
    }
    }
}
```

UniFile Object

The `UniFile` object defines and manages a data file on the server. You define the `UniFile` object through the `UniSession.open()` method. See [Using Files](#) in Chapter 2, “[Using UniObjects for Java](#),” for more information about creating and using a `UniFile` object.

UniFile() throws *UniSessionException*

This is the default constructor for this class. Do not call it directly; instead, access all files through the `UniSession.open()` method.

UniFile Object Methods

These are the methods you can use with the `UniFile` object:

-
- | | |
|--------------------------------------|--------------------------------------|
| • <code>clearFile()</code> | • <code>read()</code> |
| • <code>close()</code> | • <code>readField()</code> |
| • <code>deleteRecord()</code> | • <code>/()</code> |
| • <code>getAkInfo()</code> | • <code>setBlockingStrategy()</code> |
| • <code>getBlockingStrategy()</code> | • <code>setEncryptionType()</code> |
| • <code>getEncryptionType()</code> | • <code>setFileName()</code> |
| • <code>getFileName()</code> | • <code>setLockStrategy()</code> |
| • <code>getFileType()</code> | • <code>setRecord()</code> |
| • <code>getLockStrategy()</code> | • <code>setRecordID()</code> |
| • <code>getRecord()</code> | • <code>setReleaseStrategy()</code> |
| • <code>getRecordID()</code> | • <code>status()</code> |
| • <code>getReleaseStrategy()</code> | • <code>unlockFile()</code> |
| • <code>isOpen()</code> | • <code>unlockRecord()</code> |
| • <code>isRecordLocked()</code> | • <code>write()</code> |
| • <code>iType()</code> | • <code>writeField()</code> |
| • <code>lockFile()</code> | • <code>writeNamedField()</code> |
| • <code>lockRecord()</code> | |
| • <code>open()</code> | |
-

UniFile Object Methods



Note: All methods used with the `UniFile` object can also be used with the `UniDictionary` object.

public void clearFile () throws UniFileException

This method clears a file, deleting all its records. If the file is locked by another session or user, the current blocking strategy (as returned by the `getBlockingStrategy ()` method) determines the action to take.

This method corresponds to the `UniObjects ClearFile` method and the BASIC CLEARFILE statement.

public void close () throws UniFileException

This method closes a file. All file or record locks are released.

This method corresponds to the `UniObjects CloseFile` method and the BASIC CLOSE statement.

public void deleteRecord () throws UniFileException

public void deleteRecord (Object aRecordIDObj) throws UniFileException

public UniDataSet deleteRecord (UniDataSet aSet) throws UniFileException

This method deletes a record.

The `UniDataSet deleteRecord` method deletes a set of records defined in the `aSet` collection. It returns a dataset representing the status of each deletion, which can be examined if necessary. See “[UniDataSet Object](#)” on page 4-127 for more details.

Object `aRecordIDObj` is the record ID of the record to delete. If it is not specified, the value set by the `setRecordID ()` method is used.

This example deletes a record `rec` from the file `uFile`:

```
uFile.deleteRecord(rec);
```

This method corresponds to the UniObjects **DeleteRecord** method and the BASIC DELETE and DELETEU statements.

public UniDynArray getAkInfo (Object akNameObj) throws UniFileException

This method returns information about the secondary indexes in a UniFile object as a UniDynArray object. Value marks separate elements of the dynamic array.

Object akNameObj is the field name of the secondary index whose information you want.

The meaning of the result depends on the type of index, as follows:

- For D-type indexes: field 1 contains D as the first character and field 2 contains the location number of the indexed field.
- For I-type indexes: field 1 contains I as the first character, field 2 contains the I-type expression, and the compiled I-type code occupies fields 19 onward.
- For both D-type and I-type indexes:
 - The second value of field 1 is 1 if the index needs to be rebuilt, or an empty string otherwise.
 - The third value of field 1 is 1 if the index is null-suppressed, or an empty string otherwise.
 - The fourth value of field 1 is 1 if automatic updates are disabled, or an empty string otherwise.
 - Field 6 contains an S if the index is single-valued and an M if it is multivalued.

If akNameObj is an empty string, a list of secondary indexes on the file returns as a dynamic array of fields.

This method corresponds to the UniObjects **GetAkInfo** method and the BASIC INDICES function.

public int getBlockingStrategy()

This method returns the current blocking strategy, which is the action taken when a record or file lock blocks a database file operation. If you do not specify a value with the `setBlockingStrategy()` method, the default blocking strategy is used, which you can get through the `UniSession.getDefaultBlockingStrategy()` method.

Use the `getBlockingStrategy()` method with the `setLockStrategy()` and `setReleaseStrategy()` methods.

The blocking strategy is one of the following values:

Value	Token	Description
1	<code>UniObjectsTokens.UVT_WAIT_LOCKED</code>	If the record is locked, wait until it is released (see Note).
2	<code>UniObjectsTokens.UVT_RETURN_LOCKED</code>	Return a status value to indicate the state of the lock. This is the default value.

getBlockingStrategy Values

Note: Use the `UniObjectsTokens.UVT_WAIT_LOCKED` value with caution. While the method is waiting for the lock to be released, your client window is effectively frozen and will not respond to mouse clicks.

This method corresponds to the `UniObjects` **BlockingStrategy** property.



public int getEncryptionType()

This method returns the current encryption type. The encryption type is one of the following:

Value	Token	Description
0	UniObjectsTokens.NO_ENCRYPT	Do not encrypt data. This is the default value.
1	UniObjectsTokens.UV_ENCRYPT	Encrypt data using internal database encryption.

getEncryptionType Values

public String getFileName()

This method returns the name of the database file supplied by the `UniSession.open()` method. It corresponds to the UniObjects **FileName** property.

public int getFileType()

This method returns the file type. Valid file types are:

- 2 through 18 (static hashed files)
- 1 or 19 (nonhashed files)
- 25 (B-tree files)
- 30 (dynamic hashed files)

This method corresponds to the UniObjects **FileType** property.

public int getLockStrategy()

This method returns the current lock strategy for setting locks during read operations on the file. If no value is set, the default lock strategy value is used, which you can get through the `UniSession.getDefaultLockStrategy()` method. The lock strategy is one of the following values:

Value	Token	Description
0	<code>UniObjectsTokens.NO_LOCKS</code>	No locking. This is the default value.
1	<code>UniObjectsTokens.UVT_EXCLUSIVE_READ</code>	Sets an exclusive update lock (READU).
2	<code>UniObjectsTokens.UVT_SHARED_READ</code>	Sets a shared read lock (READL).

getLockStrategy Values

Use this method with the `setBlockingStrategy()` and `setReleaseStrategy()` methods.

public UniString getRecord()

This method returns the contents of a record as a string. You should convert this to a `UniDynArray` object in order to operate on the elements of the dynamic array. It is updated whenever a `read()`, `readField()`, or `readNamedField()` method is called.

This method corresponds to the `UniObjects` **Record** property.

public String getRecordID()

This method returns the ID of the record to be processed. It corresponds to the `UniObjects` **RecordId** property.

public int getReleaseStrategy()

This method returns the current release strategy for releasing locks set during reads and calls to the `lockRecord()` method. If no value is set, the default release strategy value is used, which you can get through the `UniSession.getDefaultReleaseStrategy()` method.

Use `getReleaseStrategy()` with the `setBlockingStrategy()` and `setLockStrategy()` methods.

`getReleaseStrategy()` returns one of the following values:

Value	Token	Description
1	<code>UniObjectsTokens.WRITE_RELEASE</code>	Releases the lock when the write finishes. This is the default value.
2	<code>UniObjectsTokens.UVT_READ_RELEASE</code>	Releases the lock after the record is read.
4	<code>UniObjectsTokens.UVT_EXPLICIT_RELEASE</code>	Maintains locks as specified by the <code>setLockStrategy()</code> method. Locks can be released only with the <code>unlockRecord()</code> method.
8	<code>UniObjectsTokens.UVT_CHANGE_RELEASE</code>	Releases the lock whenever a new value is set through the <code>setRecordID()</code> method.

getReleaseStrategy Values

All the values are additive. If you specify `UniObjectsTokens.UVT_EXPLICIT_RELEASE` with `UniObjectsTokens.WRITE_RELEASE` and `UniObjectsTokens.UVT_READ_RELEASE`, it takes a lower priority. The initial release strategy value is 12, that is, release locks when the value of the record ID changes or when locks are released explicitly.

This method corresponds to the `UniObjects` **ReleaseStrategy** property.

public boolean isOpen()

This method checks to see if a file is open. It returns `true` if the file is open, `false` if it is not. For example:

```
UniFile uFile = uSession.open("FOOBAR");
System.out.println("Opened file");
if (uFile.isOpen())
{
    System.out.println("Setting new block size");
    .
    .
    .
}
```

This method corresponds to the UniObjects **isOpen** method.

public boolean isRecordLocked() throws UniFileException

public boolean isRecordLocked (Object aRecordIDObj) throws UniFileException

This method determines whether or not a user or session has locked a specified record.

Object `aRecordIDObj` is the ID of the record you want to check. If `aRecordIDObj` is not specified, the value set by the `setRecordID()` method is used.

public UniString iType (Object aRecordIDObj, Object aITypeObj) throws UniFileException

This method evaluates the specified I-descriptor and returns the evaluated string. It applies no conversions to the data.

Object `aRecordIDObj` is the record ID of the record supplied as data.

Object `aITypeObj` is the record ID of the I-descriptor to evaluate.

This method corresponds to the UniObjects **iType** method and the BASIC `ITYPE` function.

public void lockFile() throws UniFileException

This method locks an associated database file. It does not rely on any of the locking strategies such as those returned by the `getBlockingStrategy()`, `getLockStrategy()`, or `getReleaseStrategy()` methods. If another user or session has locked the file, `lockFile()` throws `UniFileException`.

This method corresponds to the UniObjects **LockFile** method and the BASIC FILELOCK statement.

public void lockRecord (int aLockFlag) throws UniFileException

public void lockRecord (Object aRecordIDObj, int aLockFlag) throws UniFileException

public UniDataSet lockRecord (UniDataSet aSet) throws UniFileException

This method locks a record using the type of lock specified by `aLockFlag`. Use this method to override the current locking strategy. The `UniDataSet lockRecord` method locks a set of records defined by `aSet`. It returns a `UniDataSet` representing the status of each individual operation.

`int aLockFlag` is one of the following values:

Value	Token	Description
1	UniObjectsTokens.UVT_EXCLUSIVE_READ	Sets an exclusive update lock (READU).
2	UniObjectsTokens.UVT_SHARED_READ	Sets a shared read lock (READL).

aLockFlag Values

Object `aRecordIDObj` is the ID of the record you want to lock. If you do not specify `aRecordIDObj`, the record ID is the one previously set by the `setRecordID()` method.



Using this method is equivalent to calling the `read()`, `readField()`, or `readNamedField()` methods with the lock strategy set to the value of `aLockFlag`. If the value of `aLockFlag` is not valid, the method returns without performing any locking.

Note: You may need to explicitly unlock the record using the `unlockRecord()` method, depending upon the release strategy value.

This method corresponds to the UniObjects **LockRecord** method and the BASIC RECORDLOCKL and RECORDLOCKU statements.

public void open() throws UniFileException

This method opens a data file. If `open()` cannot open the file, it throws `UniFileException`.

public UniString read() throws UniFileException

public UniString read (Object aRecordIDObj) throws UniFileException

public UniString read (Object aRecordIDObj, int aLockFlag) throws UniFileException

public UniDataSet read (UniDataSet aSet) throws UniFileException

This method reads a database record.

The `UniDataSet` `read` method reads a set of records defined by `aSet`. It returns the result set as a `UniDataSet`, which can be used to get each individual result as well as the status of the operation on the record requested.

Object `aRecordIDObj` is the ID of the record you want to read. If you do not specify `aRecordIDObj`, the record ID is the one set previously by the `setRecordID()` method.

int `aLockFlag` is the locking strategy to use instead of the current default locking strategy as set by the `setLockStrategy()` method.

This example reads record 54637 in the ORDERS file:

```
UniFile uFile = uSession.open ("ORDERS");
UniString rec = uFile.read ("54637");
```

This method corresponds to the UniObjects **Read** method and the BASIC READ, READL, and READU statements.

```
public UniString readField (Object aRecordIDObj, int aFieldNumber) throws
UniFileException
```

```
public UniString readField (Object aRecordIDObj, int aFieldNumber, int aLockFlag)
throws UniFileException
```

```
public UniDataSet readField (UniDataSet aSet, String fieldList) throws
UniFileException
```

This method reads a field value from a record.

The UniDataSet readField method reads the set of fields specified by fieldList, a field-mark-delimited string denoting the number of each field requested. It reads these fields for the set of record IDs specified in aSet and returns the results as a UniDataSet object.

Object aRecordIDObj is the ID of the record whose field value you want to read. If you do not specify aRecordIDObj, the record ID is the one set previously by the setRecordID() method.

int aFieldNumber is the number of the field to read. Specify field 0 (the record ID) to check if a record exists.

int aLockFlag is the locking strategy to use instead of the current default locking strategy as set by the setLockStrategy() method.

This method corresponds to the UniObjects **ReadField** method and the BASIC READV, READVL, and READVU statements.

```
UniFile uFile = uSession.open("ORDERS");
UniString rec = uFile.readField( uFile, 2 );
);
```

public UniString readNamedField (Object aFieldNameObj) throws UniFileException

public UniString readNamedField (Object aRecordIDObj, Object aFieldNameObj) throws UniFileException

public UniDataSet readNamedField (UniDataSet aSet, String fieldList) throws UniFileException

This method reads a field value from a record, performing any output conversion defined in the file dictionary for the field.

The `UniDataSet readNamedField` method reads the set of fields specified by `fieldList`, a field-mark-delimited string denoting the number of each field requested. It reads these fields for the set of record IDs specified in `aSet` and returns the results as a `UniDataSet` object.

Object `aFieldNameObj` is the name of the field you want to read. The field must be defined by a D-descriptor or an I-descriptor in the file dictionary.

Object `aRecordIDObj` is the ID of the record whose field value you want to read. If you do not specify `aRecordIDObj`, the record ID is the one set previously by the `setRecordID()` method.



Note: This method needs to read the file dictionary in order to determine the location of the field. This can affect the performance of your application. If performance is an issue, use the `readField()` method. For more information about using the `readNamedField()` method, see [Data Conversion](#) in Chapter 2, “Using UniObjects for Java.”

If `readNamedField()` returns the error `UVE_RNF` (record not found), the missing record can be either the data record whose field value you want to read or the dictionary record defining the field.

This method corresponds to the UniObjects **ReadNamedField** method.

```
UniFile uFile = uSession.open("ORDERS");
UniString addressField = uFile.readNamedField("SMITH,J",
"ADDRESS.FIELD" );
```

***public void setBlockingStrategy (int aBlock) throws
UniFileException***

This method sets up the action taken when a database file operation is blocked by a record or file lock.

int aBlock is the blocking strategy to use, which is one of the following:

Value	Token	Description
1	UniObjectsTokens.UVT_WAIT_LOCKED	If the record is locked, wait until it is released (see Note).
2	UniObjectsTokens.UVT_RETURN_LOCKED	Return a status value to indicate the state of the lock. This is the default value.

setBlockingStrategy Values

Note: Use the UniObjectsTokens.UVT_WAIT_LOCKED value with caution. While the method is waiting for the lock to be released, your client window is effectively frozen and will not respond to mouse clicks.

This method corresponds to the UniObjects **BlockingStrategy** property.

public void setEncryptionType (int aEncryptType)

This method sets the type of encryption to use for all operations on the UniFile object.

int aEncryptType is the encryption type, which is one of the following:

Value	Token	Description
0	UniObjectsTokens.NO_ENCRYPT	Do not encrypt data. This is the default value.
1	UniObjectsTokens.UV_ENCRYPT	Encrypt data using internal database encryption.

aEncryptType Values



public void setFileName (Object aFileNameObj)

This method sets the name of the database file to open using the `open ()` method.

Object `aFileNameObj` is the name of the file to open.

This method corresponds to the UniObjects **FileName** property.

public void setLockStrategy (int aLock) throws UniFileException

This method sets up the strategy for setting locks during read operations on the file.

`int aLock` is the lock strategy you want to use, which is one of the following:

Value	Token	Description
0	UniObjectsTokens.NO_LOCKS	No locking. This is the default value.
1	UniObjectsTokens.UVT_EXCLUSIVE_READ	Sets an exclusive update lock (READU).
2	UniObjectsTokens.UVT_SHARED_READ	Sets a shared read lock (READL).

aLock Values

This method corresponds to the UniObjects **LockStrategy** property.

public void setRecord (Object aStringObj)

This method sets the values of a record.

Object `aStringObj` is the data to write to a record using subsequent write methods.

This method corresponds to the UniObjects **Record** property.

public void setRecordID (Object aStringObj) throws UniFileException

This method sets the ID of the record to process using `UniFile` object methods such as `read ()` or `write ()`.

Object `aStringObj` is the record ID to use in subsequent read and write methods.

This method corresponds to the UniObjects **RecordId** property.

public void setReleaseStrategy (int aRelease)

This method sets up the strategy for releasing locks set by the `read()`, `readField()`, and `readNamedField()` methods.

`int aRelease` is the release strategy to use, which is one of the following:

Value	Token	Description
1	<code>UniObjectsTokens.WRITE_RELEASE</code>	Releases the lock after the record is written. This is the default value.
2	<code>UniObjectsTokens.UVT_READ_RELEASE</code>	Releases the lock after the record is read.
4	<code>UniObjectsTokens.UVT_EXPLICIT_RELEASE</code>	Maintains locks as specified by the <code>setLockStrategy()</code> method. Locks can be released only with the <code>unlockRecord()</code> method.
8	<code>UniObjectsTokens.UVT_CHANGE_RELEASE</code>	Releases the lock whenever a new value is set by the <code>setRecordId()</code> method.

aRelease Values

All the values are additive. If you specify `UniObjectsTokens.UVT_EXPLICIT_RELEASE` with `UniObjectsTokens.WRITE_RELEASE` and `UniObjectsTokens.UVT_READ_RELEASE`, it takes a lower priority. The initial release strategy value is 12, that is, release locks when the value of the record ID changes or when locks are released explicitly.

This method corresponds to the UniObjects **ReleaseStrategy** property.

public int status()

This method retrieves the status code returned by each method. Refer to each method for a description of these status values.

This method corresponds to the UniObjects **Status** property.

public void unlockFile() throws UniFileException

This method removes all file locks from a database file. It corresponds to the UniObjects **UnlockFile** method and the BASIC FILEUNLOCK statement.

```
uFile.unlockFile();
```

public void unlockRecord() throws UniFileException

public void unlockRecord (Object aRecordIDObj) throws UniFileException

public UniDataSet unlockRecord (UniDataSet aSet) throws UniFileException

This method unlocks a record. The UniDataSet unlockRecord method unlocks a set of records defined by aSet. It returns a UniDataSet representing the status of each individual operation.

Object aRecordIDObj is the ID of the record you want to unlock. If you do not specify aRecordIDObj, the record ID is the one previously set by the setRecordID() method.

This method corresponds to the UniObjects **UnlockRecord** method and the BASIC RELEASE statement.

```
uFile.unlockRecord( "REC3" );
```

public void write() throws UniFileException

***public void write (Object aRecordIDObj, Object aRecordDataObj)
throws UniFileException***

***public void write (Object aRecordIDObj, Object aRecordDataObj, int
aLockFlag) throws UniFileException***

public UniDataSet write (UniDataSet aSet) throws UniFileException

This method writes data to a record.

The `UniDataSet` `write` method writes a set of records defined by `aSet`. It returns the result set as a `UniDataSet`, which can be used to get each individual result as well as the status of the operation on the record requested.

Object `aRecordIDObj` is the ID of the record to write to. If you do not specify `aRecordIDObj`, the record ID is the one previously set by the `setRecordID()` method.

Object `aRecordDataObj` is the value to write to the record. If you do not specify `aRecordDataObj`, the value to write is the one previously set by the `setRecord()` method.

`int aLockFlag` is the type of lock to use.

Call the `status()` method after executing a `write()` method to determine the state of record locking during the operation, as follows:

Value	Description
0	The record was locked before the operation.
-2	The record was not locked before the operation.

status Values

This method corresponds to the `UniObjects` **Write** method and the BASIC WRITE and WRITEU statements.

public void writeField (int aFieldNumber) throws UniFileException

***public void writeField (Object aRecordIDObj, Object
aRecordDataObj, int aFieldNumber) throws UniFileException***

***public void writeField (Object aRecordIDObj, Object
aRecordDataObj, int aFieldNumber, int aLockFlag) throws
UniFileException***

***public UniDataSet writeField (UniDataSet aSet, String fieldList)
throws UniFileException***

This method writes data to a single field in a record.

The UniDataSet writeField method writes the set of fields specified by fieldList, a field-mark-delimited string denoting the number of each field requested. It writes these fields for the set of record IDs specified in aSet and returns the results as a UniDataSet object.

int aFieldNumber is the number of the field to write to. If you do not specify aFieldNumber, field 1 is written to.

Object aRecordIDObj is the ID of the record to write to. If you do not specify aRecordIDObj, the record ID is the one previously set by the setRecordID() method.

Object aRecordDataObj is the field value to write to the record. If you do not specify aRecordDataObj, the field value to write is the one previously set by the setRecord() method.

int aLockFlag is the type of lock to use.

Call the `status()` method after executing a `write()` method to determine the state of record locking during the operation, as follows:

Value	Description
0	The record was locked before the operation.
-2	The record was not locked before the operation.
-3	A SQL Integrity check has failed.

status Values

This method corresponds to the UniObjects **WriteField** method and the BASIC WRITEV and WRITEVU statements.

This example writes the string `NewFieldValue` into field 3 of the record `REC3`.

```
uFile.writeField( "REC3", "NewFieldValue", 3 );
```

public void writeNamedField (Object aFieldName, Object aString) throws UniFileException

public void writeNamedField (Object aRecordIDObj, Object aFieldName, Object aString) throws UniFileException

public UniDataSet writeNamedField (UniDataSet aSet, String fieldList) throws UniFileException

This method writes data to a single field in a record, performing any input conversion defined in the file dictionary for the field.

The UniDataSet `writeNamedField` method writes the set of fields specified by `fieldList`, a field-mark-delimited string denoting the name of each field requested. It writes these fields for the set of record IDs specified in `aSet` and returns the results as a UniDataSet object.



Note: `writeNamedField()` *does not convert distinct values in a multivalued field.*

Object `aFieldName` is the name of the field as defined in the file dictionary.

Object `aString` is the field value to write to the record.

Object `aRecordIDObject` is the ID of the record you want to write to. If you do not specify `aRecordIDObj`, the record ID is the one previously set by the `setRecordID()` method.

This method corresponds to the UniObjects **WriteNamedField** method.

This example writes the string `NewFieldValue` into field 3 of the record `REC3`, if `FIELD3` is defined in the dictionary as field 3.

```
uFile.writeNamedField( "REC3", "NewFieldValue", "FIELD3" );
```

Example Using the UniFile Object

```
import asjava.uniobjects.*;
public class UOJtest{
    public static void main(String args[]){
        try{
            UniJava uJava = new UniJava();
            UniSession uSession = uJava.openSession();
            uSession.connect("alfa", "davem", "password", "/usr/uv/sales" );
            UniFile orderFile = uSession.open( "ORDERS" );
            UniString custRec = orderFile.read("SMITH,J");
            custRec.writeNamedField( "SMITH,J", "IOWA", "STATE.FIELD" );
        } catch (UniException e )
        {
            /* code to handle exception */
        }
    }
}
```

UniDictionary Object

The `UniDictionary` object is a specific instance of the `UniFile` object with extensions specific to file dictionaries. It defines and manages a file dictionary. The `UniDictionary` object has methods that refer to specific fields in dictionary records. For more information about file dictionaries and how to use them, see [The Database Environment](#) and [Using a Dictionary](#) in Chapter 2, “Using UniObjects for Java.” For more information about the fields in a dictionary, see *Universe System Description*.

UniDictionary()

This is the default constructor for the class. Do not call it directly; instead, access all file dictionaries through the `UniSession.openDict()` method.

UniDictionary Object Methods

These are the methods you can use with the `UniDictionary` object:

clearFile()	getSM()
close()	getSQLType()
deleteRecord()	getType()
getAkInfo()	isOpen()
getAssoc()	iType()
getBlockingStrategy()	lockFile()
getConv()	lockRecord()
getEncryptionType()	open()
getFileName()	read()
getFileType()	readField()
getFormat()	readNamedField()
getLoc()	setAssoc()
getLockStrategy()	setBlockingStrategy()
getName()	setConv()
getRecord()	setEncryptionType()
getRecordID()	setFileName()
getReleaseStrategy()	setFormat()
setLoc()	setType()
setLockStrategy()	status()
setName()	unlockFile()
setRecord()	unlockRecord()
setRecordID()	write()
setReleaseStrategy()	writeField()
setSM()	writeNamedField()
setSQLType()	

UniDictionary Objects

public void clearFile () throws UniFileException

This method clears a file, deleting all its records. If the file is locked by another session or user, the current blocking strategy (as returned by the `getBlockingStrategy ()` method) determines the action to take.

This method corresponds to the UniObjects **ClearFile** method and the BASIC CLEARFILE statement.

```
uDict.clearFile();
```

public void close () throws UniFileException

This method closes a file. All file or record locks are released.

This method corresponds to the UniObjects **CloseFile** method and the BASIC CLOSE statement.

```
uDict.close();
```

```
public void deleteRecord () throws UniFileException
```

public void deleteRecord (Object aRecordIDObj) throws UniFileException

This method deletes a record.

Object `aRecordIDObj` is the record ID of the record to delete. If it is not specified, the value set by the `setRecordID ()` method is used.

This method corresponds to the UniObjects **DeleteRecord** method and the BASIC DELETE and DELETEDU statements.

```
uDict.deleteRecord( "FOOBAR" );
```

public UniDynArray getAkInfo (Object akNameObj) throws UniFileException

This method returns information about the secondary indexes in a UniDictionary object as a UniString object. Value marks separate elements of the dynamic array.

Note: This method applies to UniVerse only. UniData does not support indexes to file dictionaries.



Object `akNameObj` is the field name of the secondary index whose information you want.

The meaning of the result depends on the type of index, as follows:

- For D-type indexes: field 1 contains D as the first character and field 2 contains the location number of the indexed field.
- For I-type indexes: field 1 contains I as the first character, field 2 contains the I-type expression, and the compiled I-type code occupies fields 19 onward.
- For both D-type and I-type indexes:
 - The second value of field 1 is 1 if the index needs to be rebuilt, or an empty string otherwise.
 - The third value of field 1 is 1 if the index is null-suppressed, or an empty string otherwise.
 - The fourth value of field 1 is 1 if automatic updates are disabled, or an empty string otherwise.
 - Field 6 contains an S if the index is single-valued and an M if it is multivalued.

If `akNameObj` is an empty string, a list of secondary indexes on the file returns as a dynamic array of fields.

This method corresponds to the UniObjects **GetAkInfo** method and the BASIC INDICES function.

```
UniDynArray akInfo = uDict.getAkinfo( akName );
```

public UniString getAssoc() throws UniFileException

public UniString getAssoc (Object aRecordID) throws UniFileException

This method returns the value in the ASSOC field (field 7) from a dictionary record.

Object `aRecordID` is the ID of the record you want.

If you do not specify `aRecordID`, the record is specified by the `setRecordID()` method.

This method corresponds to the UniObjects **ASSOC** property.

public int getBlockingStrategy()

This method returns the current blocking strategy, which is the action taken when a record or file lock blocks a database file operation. If you do not specify a value with the `setBlockingStrategy()` method, the default blocking strategy is used, which you can get through the `UniSession.getDefaultBlockingStrategy()` method.

Use the `getBlockingStrategy()` method with the `setLockStrategy()` and `setReleaseStrategy()` methods.

The blocking strategy is one of the following values:

Value	Token	Description
1	<code>UniObjectsTokens.UVT_WAIT_LOCKED</code>	If the record is locked, wait until it is released (see Note).
2	<code>UniObjectsTokens.UVT_RETURN_LOCKED</code>	Return a status value to indicate the state of the lock. This is the default value.

getBlockingStrategy Values

Note: Use the `UniObjectsTokens.UVT_WAIT_LOCKED` value with caution. While the method is waiting for the lock to be released, your client window is effectively frozen and will not respond to mouse clicks.

This method corresponds to the UniObjects **BlockingStrategy** property.

public UniString getConv() throws UniFileException

public UniString getConv (Object aRecordID) throws UniFileException

This method returns the value in the CONV field (field 3) from a dictionary record.

Object `aRecordID` is the ID of the record you want.



If you do not specify `aRecordID`, the record is specified by the `setRecordID()` method.

For information about conversion codes, see *UniVerse BASIC*.

This method corresponds to the UniObjects **CONV** property.

public int getEncryptionType()

This method returns the current encryption type. The encryption type is one of the following:

Value	Token	Description
0	UniObjectsTokens.NO_ENCRYPT	Do not encrypt data. This is the default value.
1	UniObjectsTokens.UV_ENCRYPT	Encrypt data using internal database encryption.

getEncryptionType Values

public UniString getFileName()

This method returns the name of the database file supplied by the `UniSession.open()` method. It corresponds to the UniObjects **FileName** property.

public int getFileType()

This method returns the file type. Valid file types are:

- 2 through 18 (static hashed files)
- 1 or 19 (nonhashed files)
- 25 (B-tree files)
- 30 (dynamic hashed files)

This method corresponds to the UniObjects **FileType** property.

public UniString getFormat() throws UniFileException

public UniString getFormat (Object aRecordID) throws UniFileException

This method returns the value in the FORMAT field (field 5) from a dictionary record.

Object `aRecordID` is the ID of the record you want.

If you do not specify `aRecordID`, the record is specified by the `setRecordID()` method.

This method corresponds to the UniObjects **FORMAT** property.

public UniString getLoc() throws UniFileException

public UniString getLoc (Object aRecordID) throws UniFileException

This method returns the value in the LOC field (field 2) from a dictionary record.

Object `aRecordID` is the ID of the record you want.

If you do not specify `aRecordID`, the record is specified by the `setRecordID()` method.

This method corresponds to the UniObjects **LOC** property.

public int getLockStrategy()

This method returns the current lock strategy for setting locks during read operations on the file. If no value is set, the default lock strategy value is used, which you can get through the `UniSession.getDefaultLockStrategy()` method. The lock strategy is one of the following values:

Value	Token	Description
0	<code>UniObjectsTokens.NO_LOCKS</code>	No locking. This is the default value.
1	<code>UniObjectsTokens.UVT_EXCLUSIVE_READ</code>	Sets an exclusive update lock (READU).
2	<code>UniObjectsTokens.UVT_SHARED_READ</code>	Sets a shared read lock (READL).

getDefaultLockStrategy Values

Use this method with the `setBlockingStrategy()` and `setReleaseStrategy()` methods.

public UniString getName() throws UniFileException

public UniString getName (Object aRecordID) throws UniFileException

This method returns the value in the NAME field (field 4) from a dictionary record.

Object `aRecordID` is the ID of the record you want.

If you do not specify `aRecordID`, the record is specified by the `setRecordID()` method.

This method corresponds to the UniObjects **NAME** property.

public UniString getRecord()

This method returns the contents of a record as a string. You should convert this to a `UniDynArray` object in order to operate on the elements of the dynamic array. It is updated whenever a `read()`, `readField()`, or `readNamedField()` method is called.

This method corresponds to the UniObjects **Record** property.

public String getRecordID()

This method returns the ID of the record to be processed. It corresponds to the UniObjects **RecordId** property.

public int getReleaseStrategy()

This method returns the current release strategy for releasing locks set during reads and calls to the `lockRecord()` method. If no value is set, the default release strategy value is used, which you can get through the `UniSession.getDefaultReleaseStrategy()` method.

Use `getReleaseStrategy()` with the `setBlockingStrategy()` and `setLockStrategy()` methods.

`getReleaseStrategy()` returns one of the following values:

Value	Token	Description
1	<code>UniObjectsTokens.WRITE_RELEASE</code>	Releases the lock when the write finishes. This is the default value.
2	<code>UniObjectsTokens.UVT_READ_RELEASE</code>	Releases the lock after the record is read.
4	<code>UniObjectsTokens.UVT_EXPLICIT_RELEASE</code>	Maintains locks as specified by the <code>setLockStrategy()</code> method. Locks can be released only with the <code>unlockRecord()</code> method.
8	<code>UniObjectsTokens.UVT_CHANGE_RELEASE</code>	Releases the lock whenever a new value is set through the <code>setRecordID()</code> method.

getReleaseStrategy Values

All the values are additive. If you specify `UniObjectsTokens.UVT_EXPLICIT_RELEASE` with `UniObjectsTokens.WRITE_RELEASE` and `UniObjectsTokens.UVT_READ_RELEASE`, it takes a lower priority. The initial release strategy value is 12, that is, release locks when the value of the record ID changes or when locks are released explicitly.

This method corresponds to the `UniObjects` **ReleaseStrategy** property.

public UniString getSM() throws UniFileException

public UniString getSM (Object aRecordID) throws UniFileException

This method returns the value in the SM field (field 6) from a dictionary record.

Object `aRecordID` is the ID of the record you want.

If you do not specify `aRecordID`, the record is specified by the `setRecordID()` method.

This method corresponds to the `UniObjects` **SM** property.

public UniString getSQLType() throws UniFileException

public UniString getSQLType (Object aRecordID) throws UniFileException

This method returns the value in the DATATYPE field (field 8) from a dictionary record.

Note: This method applies only to `UniVerse`.

Object `aRecordID` is the ID of the record you want.

If you do not specify `aRecordID`, the record is specified by the `setRecordID()` method.

This method corresponds to the `UniObjects` **SQLTYPE** property.



public UniString getType() throws UniFileException

public UniString getType (Object aRecordID) throws UniFileException

This method returns the value in the CODE field (field 1) from the dictionary record.

Object `aRecordID` is the ID of the record you want.

If you do not specify `aRecordID`, the record is specified by the `setRecordID()` method.

The first characters of the CODE field indicate the type of field the dictionary record is defining. Valid types are:

D	D-descriptor
I	I-descriptor
V	(UniData only) V-descriptor
PH	Phrase
X	(UniVerse only) X-descriptor

This method corresponds to the UniObjects **TYPE** property.

public boolean isOpen()

This method checks to see if a file is open. It returns `true` if the file is open, `false` if it is not.

This method corresponds to the UniObjects **IsOpen** method.

public boolean isRecordLocked() throws UniFileException

public boolean isRecordLocked (Object aRecordIDObj) throws UniFileException

This method determines whether or not a user or session has locked a specified record.

Object `aRecordIDObj` is the ID of the record you want to check. If `aRecordIDObj` is not specified, the value set by the `setRecordID()` method is used.

***public UniString iType (Object aRecordIDObj, Object aITypeObj)
throws UniFileException***

This method evaluates the specified I-descriptor and returns the evaluated string. It applies no conversions to the data.

Object `aRecordIDObj` is the record ID of the record supplied as data.

Object `aITypeObj` is the record ID of the I-descriptor to evaluate.

This method corresponds to the UniObjects **IType** method and the BASIC ITYPE function.

public void lockFile () throws UniFileException

This method locks an associated database file. It does not rely on any of the locking strategies such as those returned by the `getBlockingStrategy()`, `getLockStrategy()`, or `getReleaseStrategy()` methods. If another user or session has locked the file, `lockFile()` throws `UniFileException`.

This method corresponds to the UniObjects **LockFile** method and the BASIC FILELOCK statement.

```
uDict.lockFile();
```

public void lockRecord (int aLockFlag) throws UniFileException

***public void lockRecord (Object aRecordIDObj, int aLockFlag)
throws UniFileException***

This method locks a record using the type of lock specified by `aLockFlag`. Use this method to override the current locking strategy.

int aLockFlag is one of the following values:

Value	Token	Description
1	UniObjectsTokens.UVT_EXCLUSIVE_READ	Sets an exclusive update lock (READU).
2	UniObjectsTokens.UVT_SHARED_READ	Sets a shared read lock (READL).

aLockFlag Values

Object aRecordIDObj is the ID of the record you want to lock. If you do not specify aRecordIDObj, the record ID is the one previously set by the setRecordID() method.

Using this method is equivalent to calling the read(), readField(), or readNamedField() methods with the lock strategy set to the value of aLockFlag. If the value of aLockFlag is not valid, the method returns without performing any locking.



***Note:** You may need to explicitly unlock the record using the unlockRecord() method, depending upon the release strategy value.*

This method corresponds to the UniObjects **LockRecord** method and the BASIC RECORDLOCKL and RECORDLOCKU statements.

```
uDict.lockRecord( "FOOBAR", UniObjectsTokens.UVT_EXCLUSIVE_READ );
```

public void open() throws UniFileException

This method opens a database data file. If open() cannot open the file, it throws UniFileException.

public UniString read() throws UniFileException

public UniString read (Object aRecordIDObj) throws UniFileException

public UniString read (Object aRecordIDObj, int aLockFlag) throws UniFileException

This method reads a database record.

Object `aRecordIDObj` is the ID of the record you want to read. If you do not specify `aRecordIDObj`, the record ID is the one set previously by the `setRecordID()` method.

`int aLockFlag` is the locking strategy to use instead of the current default locking strategy as set by the `setLockStrategy()` method.

This method corresponds to the UniObjects **Read** method and the BASIC READ, READL, and READU statements.

```
UniString orderRec = orderFile.read( "TOTALS" );
```

public UniString readField (Object aRecordIDObj, int aFieldNumber) throws UniFileException

public UniString readField (Object aRecordIDObj, int aFieldNumber, int aLockFlag) throws UniFileException

This method reads a field value from a record.

Object `aRecordIDObj` is the ID of the record whose field value you want to read. If you do not specify `aRecordIDObj`, the record ID is the one set previously by the `setRecordID()` method.

`int aFieldNumber` is the number of the field to read. Specify field 0 (the record ID) to check if a record exists.

`int aLockFlag` is the locking strategy to use instead of the current default locking strategy as set by the `setLockStrategy()` method.

This method corresponds to the UniObjects **ReadField** method and the BASIC READV, READVL, and READVU statements.

```
UniString homeAddress = orderFile.readField( "SMITH", 1 );
```

public UniString readNamedField (Object aFieldNameObj) throws UniFileException

public UniString readNamedField (Object aRecordIDObj, Object aFieldNameObj) throws UniFileException

This method reads a field value from a record, performing any output conversion defined in the file dictionary for the field.

Object `aFieldNameObj` is the name of the field you want to read. The field must be defined by a D-descriptor or an I-descriptor in the file dictionary.

Object `aRecordIDObj` is the ID of the record whose field value you want to read. If you do not specify `aRecordIDObj`, the record ID is the one set previously by the `setRecordID()` method.



Note: This method needs to read the file dictionary in order to determine the location of the field. This can affect the performance of your application. If performance is an issue, use the `readField()` method. For more information about using the `readNamedField()` method, see [Data Conversion in Chapter 2, “Using UniObjects for Java.”](#)

If `readNamedField()` returns the error UVE_RNF (record not found), the missing record can be either the data record whose field value you want to read or the dictionary record defining the field.

This method corresponds to the UniObjects **ReadNamedField** method.

```
UniString homeAddress = addressFile.readNamedField( "SMITH",  
                                                    "HOMEADDR" );
```

public void setAssoc (Object aString) throws UniFileException

public void setAssoc (Object aRecordID, Object aString) throws UniFileException

This method sets the value of the ASSOC field (field 7) of a dictionary record.

Object aString is the value to write to the ASSOC field.

Object aRecordID is the ID of the record you want.

If you do not specify aRecordID, the record is specified by the setRecordID () method.

This method corresponds to the UniObjects ASSOC property.

public void setBlockingStrategy (int aBlock) throws UniFileException

This method sets up the action taken when a database file operation is blocked by a record or file lock.

int aBlock is the blocking strategy to use, which is one of the following:

Value	Token	Description
1	UniObjectsTokens.UVT_WAIT_LOCKED	If the record is locked, wait until it is released (see Note).
2	UniObjectsTokens.UVT_RETURN_LOCKED	Return a status value to indicate the state of the lock. This is the default value.

aBlock Values

Note: Use the UniObjectsTokens.UVT_WAIT_LOCKED value with caution. While the method is waiting for the lock to be released, your client window is effectively frozen and will not respond to mouse clicks.

This method corresponds to the UniObjects **BlockingStrategy** property.



public void setConv (Object aString) throws UniFileException

public void setConv (Object aRecordID, Object aString) throws UniFileException

This method sets the value of the CONV field (field 3) of a dictionary record.

Object `aString` is the value to write to the CONV field.

Object `aRecordID` is the ID of the record you want.

If you do not specify `aRecordID`, the record is specified by the `setRecordID()` method.

For information about conversion codes, see *UniVerse BASIC*.

This method corresponds to the UniObjects **CONV** property.

public void setEncryptionType (int aEncryptType)

This method sets the type of encryption to use for all operations on the UniDictionary object.

int `aEncryptType` is the encryption type, which is one of the following:

Value	Token	Description
0	UniObjectsTokens.NO_ENCRYPT	Do not encrypt data. This is the default value.
1	UniObjectsTokens.UV_ENCRYPT	Encrypt data using internal database encryption.

aEncryptType Values

public void setFileName (Object aFileNameObj)

This method sets the name of the database file to open using the `open()` method.

Object `aFileNameObj` is the name of the file to open.

This method corresponds to the UniObjects **FileName** property.

public void setFormat (Object aString) throws UniFileException

public void setFormat (Object aRecordID, Object aString) throws UniFileException

This method sets the value of the FORMAT field (field 5) of a dictionary record.

Object `aString` is the value to write to the FORMAT field.

Object `aRecordID` is the ID of the record you want.

If you do not specify `aRecordID`, the record is specified by the `setRecordID()` method.

This method corresponds to the UniObjects **FORMAT** property.

public void setLoc (Object aString) throws UniFileException

public void setLoc (Object aRecordID, Object aString) throws UniFileException

This method sets the value of the LOC field (field 2) of a dictionary record.

Object `aString` is the value to write to the LOC field.

Object `aRecordID` is the ID of the record you want.

If you do not specify `aRecordID`, the record is specified by the `setRecordID()` method.

This method corresponds to the UniObjects **LOC** property.

public void setLockStrategy (int aLock) throws UniFileException

This method sets up the strategy for setting locks during read operations on the file.

int aLock is the lock strategy you want to use, which is one of the following:

Value	Token	Description
0	UniObjectsTokens.NO_LOCKS	No locking. This is the default value.
1	UniObjectsTokens.UVT_EXCLUSIVE_READ	Sets an exclusive update lock (READU).
2	UniObjectsTokens.UVT_SHARED_READ	Sets a shared read lock (READL).

aLock Values

This method corresponds to the UniObjects **LockStrategy** property.

public void setName (Object aString) throws UniFileException

public void setName (Object aRecordID, Object aString) throws UniFileException

This method sets the value of the NAME field (field 4) of a dictionary record.

Object aString is the value to write to the NAME field.

Object aRecordID is the ID of the record you want.

If you do not specify aRecordID, the record is specified by the setRecordID () method.

This method corresponds to the UniObjects **NAME** property.

public void setRecord (Object aStringObj)

This method sets the values of a record.

Object aStringObj is the data to write to a record using subsequent write methods.

This method corresponds to the UniObjects **Record** property.

public void setRecordID (Object aStringObj) throws UniFileException

This method sets the ID of the record to process using UniDictionary object methods such as read() or write().

Object aStringObj is the record ID to use in subsequent read and write methods.

This method corresponds to the UniObjects **RecordId** property.

public void setReleaseStrategy (int aRelease)

This method sets up the strategy for releasing locks set by the read(), readField(), and readNamedField() methods.

int aRelease is the release strategy to use, which is one of the following:

Value	Token	Description
1	UniObjectsTokens.WRITE_RELEASE	Releases the lock after the record is written. This is the default value.
2	UniObjectsTokens.UVT_READ_RELEASE	Releases the lock after the record is read.
4	UniObjectsTokens.UVT_EXPLICIT_RELEASE	Maintains locks as specified by the setLockStrategy() method. Locks can be released only with the unlockRecord() method.
8	UniObjectsTokens.UVT_CHANGE_RELEASE	Releases the lock whenever a new value is set by the setRecordId() method.

aRelease Values

All the values are additive. If you specify `UniObjectsTokens.UVT_EXPLICIT_RELEASE` with `UniObjectsTokens.WRITE_RELEASE` and `UniObjectsTokens.UVT_READ_RELEASE`, it takes a lower priority. The initial release strategy value is 12, that is, release locks when the value of the record ID changes or when locks are released explicitly.

This method corresponds to the `UniObjects` **ReleaseStrategy** property.

public void setSM (Object aString) throws UniFileException

public void setSM (Object aRecordID, Object aString) throws UniFileException

This method sets the value of an SM field (field 6) of a dictionary record.

`Object aString` is the value to write to the SM field.

`Object aRecordID` is the ID of the record you want.

If you do not specify `aRecordID`, the record is specified by the `setRecordID()` method.

This method corresponds to the `UniObjects` **SM** property.

public void setSQLType (Object aString) throws UniFileException

public void setSQLType (Object aRecordID, Object aString) throws UniFileException

This method sets the value of the DATATYPE field (field 8) of a dictionary record.

`Object aString` is the value to write to the DATATYPE field.

`Object aRecordID` is the ID of the record you want.

If you do not specify `aRecordID`, the record is specified by the `setRecordID()` method.

This method corresponds to the `UniObjects` **SQLTYPE** property.

public void setType (Object aString) throws UniFileException

public void setType (Object aRecordID, Object aString) throws UniFileException

This method sets the value of the CODE field (field 1) of the dictionary record.

Object `aString` is the value to write to the CODE field. The first characters of the CODE field indicate the type of field the dictionary record is defining. Valid types are:

D	D-descriptor
I	I-descriptor
V	(UniData only) V-descriptor
PH	Phrase
X	(UniVerse only) X-descriptor

Object `aRecordID` is the ID of the record you want.

If you do not specify `aRecordID`, the record is specified by the `setRecordID()` method.

This method corresponds to the UniObjects **TYPE** property.

public int status ()

This method retrieves the status code returned by each method. Refer to each method for a description of these status values.

This method corresponds to the UniObjects **Status** property.

public void unlockFile () throws UniFileException

This method removes all file locks from a database file. It corresponds to the UniObjects **UnlockFile** method and the BASIC FILEUNLOCK statement.

```
uDict.unlockFile();
```

```
public void unlockRecord() throws UniFileException
```

```
public void unlockRecord (Object aRecordIDObj) throws UniFileException
```

This method unlocks a record.

Object `aRecordIDObj` is the ID of the record you want to unlock. If you do not specify `aRecordIDObj`, the record ID is the one previously set by the `setRecordID()` method.

This method corresponds to the UniObjects **UnlockRecord** method and the BASIC RELEASE statement.

```
uDict.unlockRecord("FOOBAR");
```

public void write() throws UniFileException

public void write (Object aRecordIDObj, Object aRecordDataObj) throws UniFileException

public void write (Object aRecordIDObj, Object aRecordDataObj, int aLockFlag) throws UniFileException

This method writes data to a record.

Object `aRecordIDObj` is the ID of the record to write to. If you do not specify `aRecordIDObj`, the record ID is the one previously set by the `setRecordID()` method.

Object `aRecordDataObj` is the value to write to the record. If you do not specify `aRecordDataObj`, the value to write is the one previously set by the `setRecord()` method.

int `aLockFlag` is the type of lock to use.

Call the `status()` method after executing a `write()` method to determine the state of record locking during the operation, as follows:

Value	Description
0	The record was locked before the operation.
-2	The record was not locked before the operation.
-3	An SQL security check has failed.

status Values

This method corresponds to the UniObjects **Write** method and the BASIC WRITE and WRITEU statements.

```
addressFile.write( "SMITH", "10 Wayside Inn Rd." );
```

public void writeField (int aFieldNumber) throws UniFileException

public void writeField (Object aRecordIDObj, Object aRecordDataObj, int aFieldNumber) throws UniFileException

public void writeField (Object aRecordIDObj, Object aRecordDataObj, int aFieldNumber, int aLockFlag) throws UniFileException

This method writes data to a single field in a record.

Object aRecordIDObj is the ID of the record to write to. If you do not specify aRecordIDObj, the record ID is the one previously set by the setRecordID() method.

Object aRecordDataObj is the field value to write to the record. If you do not specify aRecordDataObj, the field value to write is the one previously set by the setRecord() method.

int aFieldNumber is the number of the field to write to. If you do not specify aFieldNumber, field 1 is written to.

int aLockFlag is the type of lock to use.

Call the status() method after executing a write() method to determine the state of record locking during the operation, as follows:

Value	Description
0	The record was locked before the operation.
-2	The record was not locked before the operation.
-3	An SQL security check has failed.

status Values

This method corresponds to the UniObjects **WriteField** method and the BASIC WRITEV and WRITEVU statements.

```
addressFile.writeField("SMITH", "Iowa", 4 );
```

***public void writeNamedField (Object aFieldName, Object aString)
throws UniFileException***

***public void writeNamedField (Object aRecordIDObj, Object
aFieldName, Object aString) throws UniFileException***

This method writes data to a single field in a record, performing any input conversion defined in the file dictionary for the field.



Note: *writeNamedField() does not convert distinct values in a multivalued field.*

Object `aFieldName` is the name of the field as defined in the file dictionary.

Object `aString` is the field value to write to the record.

Object `aRecordIDObject` is the ID of the record you want to write to. If you do not specify `aRecordIDObj`, the record ID is the one previously set by the `setRecordID()` method.

This method corresponds to the UniObjects **WriteNamedField** method.

```
addressFile.writeNamedField( "SMITH", "Iowa", "STATEFIELD" );
```

Example Using the UniDictionary Object

```
UniDictionary uFile = uSession.openDict("FOOBAR");  
uFile.setRecordID("DTMTEST");  
  
System.out.println("Dictionaries entries for " +  
    uFile.getRecordID());  
UniString uvs = uFile.read();  
System.out.println("DataValue = " + uFile.getRecord());  
System.out.println("Assoc = " + uFile.getAssoc());  
System.out.println("Conversion = " + uFile.getConv());  
System.out.println("Format = " + uFile.getFormat());  
System.out.println("Loc = " + uFile.getLoc());  
System.out.println("Name = " + uFile.getName());  
System.out.println("SM = " + uFile.getSM());  
System.out.println("SQLTYPE = " + uFile.getSQLType());
```

```
System.out.println("Type = " + uFile.getType());

System.out.println("uvs = " + uvs);
System.out.println();
System.out.println("Closing session to alfa ");
uvjava.closeSession(uSession);

}
catch (UniSessionException e)
{
    System.out.println("UniSessionError: MessageID=" +
        e.getErrorCode());
    System.out.println("UniSessionError: MessageText=" +
        e.getMessage());
}
catch (UniFileException e)
{
    System.out.println("UniSessionError: MessageID=" +
        e.getErrorCode());
    System.out.println("UniSessionError: MessageText=" +
        e.getMessage());
}
```

UniSequentialFile Object

The `UniSequentialFile` object defines and manages files that are processed sequentially. A sequential file is an operating system file on the server containing text or binary data that you want to use in your application. In UniVerse, sequential files are defined as type 1 or type 19 files.

For more information about using the `UniSequentialFile` object, see [Using Binary and Text Files](#) in Chapter 2, “Using UniObjects for Java.” For a program example that uses the `UniSequentialFile` object, see “[Example Using the UniSequentialFile Object](#)” on page 4-101.

UniSequentialFile()

This is the default constructor for this class. Do not call it directly; instead, create a `UniSequentialFile` object through the `UniSession.open()` method.

UniSequentialFile Object Methods

These are the methods you can use with the `UniSequentialFile` object:

<code>close()</code>	<code>readLine()</code>
<code>fileSeek()</code>	<code>setEncryptionType()</code>
<code>getEncryptionType()</code>	<code>setReadSize()</code>
<code>getReadSize()</code>	<code>setTimeout()</code>
<code>getTimeout()</code>	<code>status()</code>
<code>isOpen()</code>	<code>writeBlk()</code>
<code>open()</code>	<code>writeEOF()</code>
<code>readBlk()</code>	<code>writeLine()</code>

UniSequentialFile Object Methods

public void close() throws UniSequentialFileException

This method closes a sequential file. It corresponds to the UniObjects `CloseSeqFile` method and the BASIC `CLOSESEQ` statement.

***public void fileSeek (int aRelPos, int aOffset) throws
UniSequentialFileException***

This method moves the sequential file pointer by an offset specified in bytes relative to the current position, to the beginning of the file, or to the end of the file.

int aRelPos is the pointer's relative position in a file. Possible values are:

Value	Token	Description
0	UniObjectsTokens.UVT_START	The start of the file.
1	UniObjectsTokens.UVT_CURR	The current position.
2	UniObjectsTokens.UVT_END	The end of the file.

aRelPos Values

int aOffset is the number of bytes before or after aRelPos. A negative offset moves the pointer to a position before aRelPos.

For example:

```
byte bArray[] = new byte[128];  
uSeqnew.fileSeek(0,0);  
uSeqnew.writeEOF();  
System.in.read(bArray);  
uSeq.fileSeek(0,0);/* Position back to beginning of file */
```

This method corresponds to the UniObjects **FileSeek** method and the BASIC SEEK statement.

public void getEncryptionType ()

This method returns the current encryption type, which is one of the following:

Value	Token	Description
0	UniObjectsTokens.NO_ENCRYPT	Do not encrypt data. This is the default value.
1	UniObjectsTokens.UV_ENCRYPT	Encrypt data using internal database encryption.

getEncryptionType Values

public int getReadSize()

This method returns the number of bytes to read for each successive call to the `readBlk()` method. The read-size value is initially set to 0, which indicates that all the data should be read in a single block. When the `readBlk()` method finishes, the read-size value is reset to the number of bytes that were actually read.

This method corresponds to the UniObjects **ReadSize** property.

public int getTimeout()

This method returns the current timeout value, in seconds, for `readBlk()` operations. The default value is 0, which means no timeout.

This property corresponds to the UniObjects **Timeout** property.

public boolean isOpen()

This method checks to see if a sequential file is open. It returns `true` if the file is open, `false` otherwise. It corresponds to the UniObjects **IsOpen** method.

public void open() throws UniSequentialFileException

This method opens a file for sequential processing. It corresponds to the UniObjects **Open** method and the BASIC `OPENSEQ` statement.

public UniString readBlk() throws UniSequentialFileException

This method reads a block of data from a sequential file. The size of the data block is specified by the `setReadSize()` method.

When the `readBlk()` method finishes, the read-size value is the size of the returned string in bytes, or 0 if the method failed.

readBlk () returns the following status values:

Value	Description
-1	The file is not open for a read.
0	The read was successful.
1	The end of the file was reached.

readBlk status Values

For example:

```
uSeq.setReadSize( 4096 );
UniString rec = uSeq.readBlk();
System.out.println( "Number of bytes read " + uSeq.getReadSize()
);
System.out.println( "Status from readblk " + uSeq.status() );
```

This method corresponds to the UniObjects **ReadBlk** method and the BASIC READBLK statement.

public UniString readLine() throws UniSequentialFileException

This method reads successive lines of data from the current position in a sequential file. The lines must be delimited by an end-of-line character such as a carriage return.

readLine () returns the following status values:

Value	Description
-1	The file is not open for a read.
0	The read was successful.
1	The end of the file was reached, or the read-size value is 0 or less.

readLine status Values

For example:

```
uvstr = uSeq.readLine();
int uvstat = uSeq.status();

while ( uvstat == 0 )
{
    uvstr = uSeq.readLine();
    uSeqnew.writeLine(uvstr);
    uvstat = uSeq.status();
    System.gc();
}
```

This method corresponds to the UniObjects **ReadLine** method and the BASIC READSEQ statement.

public void setEncryptionType (int aEncryptType)

This method sets the type of encryption to use for all operations on the UniSequentialFile object.

int aEncryptType is the encryption type, which is one of the following:

Value	Token	Description
0	UniObjectsTokens.NO_ENCRYPT	Do not encrypt data. This is the default value.
1	UniObjectsTokens.UV_ENCRYPT	Encrypt data using internal database encryption.

aEncryptType Values

public void setReadSize (int aBlockSize)

This method specifies the number of bytes to read for each successive call to the readBlk () method.

int aBlockSize is the number of bytes to read in one operation.

aBlockSize is initially set to 0, which indicates that all data should be read in a single block. Set the value to a suitable number of bytes for the memory available to your application. Values less than 0 are treated as 0.

Warning: *If the value is set to 0 and there is not enough memory to hold all the data, a run-time exception occurs.*





When the `setReadBlk()` method finishes, `aBlockSize` value is reset to the number of bytes that were actually read. 0 indicates an error or the end of the file.

***Note:** Use the `setReadSize()` method before each use of the `readBlk()` method because the read-size value may have been previously modified.*

***public void setTimeout (int aTimeOut) throws
UniSequentialFileException***

This method sets the timeout for `readBlk()` operations.

`int aTimeOut` is the number of seconds to wait before timing out. The default timeout is 0.

This method corresponds to the UniObjects **Timeout** property.

public int status ()

This method retrieves the status code returned by each method. Refer to each method for a description of returned status values.

This method corresponds to the UniObjects **Status** property.

***public void writeBlk (Object aString) throws
UniSequentialFileException***

This method writes successive blocks of data at the current position in a sequential file.

Object `aString` is the block of data to write to the file.

For example:

```
UniSequentialFile uSeqOld = uSession.openSeq("BP", "OLDTEST" );
UniString rec = uSeqOld.readBlk();
UniSequentialFile uSeq = uSession.openSeq("BP", "TEST.JAVA2",
    & true);
uSeq.writeBlk(rec);
```

This method corresponds to the UniObjects **WriteBlk** method and the BASIC WRITEBLK statement.

public void writeEOF() throws UniSequentialFileException

This method writes an end-of-file marker at the current position in the sequential file. This lets a file be truncated at a specified point when used with the `fileSeek()` method.

This method corresponds to the UniObjects **WriteEOF** method and the BASIC **WEOFSEQ** statement.

For example:

```
byte bArray[] = new byte[128];
uSeq.fileSeek(0,0);
uSeq.writeEOF();
System.in.read(bArray);
uSeq.fileSeek(0,0);/* Position back to beginning of file */
```

public void writeLine (Object aString) throws UniSequentialFileException

This method writes successive lines of data at the current position in the sequential file.

Object `aString` is a line of data to write to the file.

This method corresponds to the UniObjects **WriteLine** method and the BASIC **WRITESEQ** statement.

```
while ( uvstat == 0 )
{
    uvstr = uSeq.readLine();
    uSeqnew.writeLine( uvstr );
    uvstat = uSeq.status();
    System.gc();
}
```

Example Using the UniSequentialFile Object

```
UniSequentialFile uSeq = uSession.openSeq( "BP", "TEST2",
    & false );

System.out.println( "Opened file" );
if ( uSeq.isOpen() )
{

    System.out.println( "Setting new block size" );
    uSeq.setReadSize( 4096 );
```

```

System.out.println( "Reading blk from file " );
UniString uvstr = uSeq.readBlk();
System.out.println( "Displaying blk from file" );
System.out.println( "Number of bytes read " +
    å uSeq.getReadSize() );
System.out.println( "Status from readblk " +
    å uSeq.status() );

System.out.println( uvstr );
// Let's open up a new sequential file
UniSequentialFile uSeqnew = uSession.openSeq( "BP",
    å "TEST.JAVA2", true );
uSeqnew.writeBlk( uvstr );

byte bArray[] = new byte[128];
uSeqnew.fileSeek( 0,0 );
uSeqnew.writeEOF();

System.out.print("Press Return to continue");
System.in.read(bArray);
System.out.println();
System.out.println( "Ok, let's reset the filepointer
    å back to 0" );
uSeq.fileSeek( 0,0 );/* Position back to beginning of
    å file */
System.out.println( "Let's read a line at time" );

uvstr = uSeq.readLine();
System.out.println( "First line = " + uvstr );
int uvstat = uSeq.status();
System.out.println( "Status after first read = " +
    å uvstat );

while ( uvstat == 0 )
{
    uvstr = uSeq.readLine();
    uSeqnew.writeLine( uvstr );
    uvstat = uSeq.status();
    System.out.println( "Line = " + uvstr );
    System.gc();
}
System.out.println( "Final Status = " + uvstat );
uSeq.close();
uSeqnew.close();

}

System.out.println( "Closing session " );
uvjava.closeSession( uSession );

}
catch (UniSessionException e)
{
    System.out.println("UniSessionError: MessageID=" +

```

```
        å e.getErrorCode() );
        System.out.println("UniSessionError: MessageText=" +
        å e.getMessage() );
    }
    catch (UniSequentialFileException e )
    {
        System.out.println("UniFileError: MessageID=" +
        å e.getErrorCode() );
        System.out.println("UnifileError: MessageText=" +
        å e.getMessage() );
    }
    catch (IOException e )
```

UniString Object

The `UniString` object represents `UniVerse` or `UniData` data. It contains many of the string manipulation routines users typically expect in their applications. The `UniString` class extends the general `String` and `StringBuffer` classes with numerous database-specific additions. Unlike the `String` and `StringBuffer` classes, however, operations performed on a `UniString` object change the internal data element and do not return a new object.

UniString Object Constructors

UniString()

This is the default constructor for the class. It constructs a `UniString` object with no characters in it, and with an initial capacity of 16 characters.

UniString (int aLength)

This constructs a `UniString` object with no characters in it, and with an initial capacity specified by `aLength`.

`int aLength` specifies the capacity of the string buffer.

UniString (Object aString)

This constructs a `UniString` object with the value set to `aString`.

`Object aString` is the string value to set.

public UniString (UniSession aSession)

This constructs a `UniString` object with no characters in it, and with an initial capacity of 16 characters.

`UniSession aSession` is the `UniSession` object to bind the string to.

The `UniString` object is bound to the specified `UniSession` object, which causes it to perform certain operations on the server.

UniString (UniSession aSession, Object aString)

This constructs a `UniString` object with no characters in it, and with an initial capacity of 16 characters.

`UniSession aSession` is the `UniSession` object to bind the string to.

`Object aString` is the string value to set.

UniString Object Methods

These are the methods you can use with the `UniString` object:

- `alpha ()`
- `append ()`
- `change ()`
- `charAt ()`
- `compareTo ()`
- `convert ()`
- `count ()`
- `dcount ()`
- `equals ()`
- `equalsIgnoreCase ()`
- `getBytes ()`
- `getChars ()`
- `getMarkCharacter ()`
- `iconv ()`
- `insert ()`
- `oconv ()`
- `quote ()`
- `setValue ()`
- `status ()`
- `substring ()`
- `toCharArray ()`
- `toLowerCase ()`
- `toString ()`
- `toUpperCase ()`

public boolean alpha() throws UniStringException

public boolean alpha (UniSession aSession) throws UniStringException

This method determines whether the contents of the string buffer is an alphabetic or a nonalphabetic string.

`UniSession aSession` is a `UniSession` object representing the server to contact to determine whether the string is alphabetic.

public void append (Object aString)

public void append (char str[])

public void append (char str[], int offset, int length)

public void append (boolean b)

public void append (char c)

public void append (int i)

public void append (long l)

public void append (float f)

public void append (double d)

This method appends a string to the string buffer.

Object `aString` is an object whose value is converted to a string.

char `str[]` is a character string.

int `offset` is the index of the first character to append.

int `length` is the number of characters to append.

boolean `b` is a Boolean value, converted to a string.

char `c` is a character value.

int `i` is an integer value, converted to a string.

long `l` is a long numeric value, converted to a string.

float `f` is a floating decimal value, converted to a string.

double `d` is a double numeric value, converted to a string.

public void change (Object aSubString, Object aReplacementString)

public void change (Object aSubString, Object aReplacementString, int aOccurrence)

public void change (Object aSubString, Object aReplacementString, int aOccurrence, int aStart)

This method replaces aSubString with aReplacementString.

Object aSubString is the current string.

Object aReplacementString is the replacement string for aSubString.

int aOccurrence is an integer specifying which occurrence of aSubString to change.

int aStart is an integer specifying the first occurrence to replace. If aStart is less than 1, the replacement starts from the beginning.

public char charAt (int index) throws UniStringException

This method returns the character at the index specified by index.

int index is a number greater than or equal to 0, specifying the index of the character you want. index must be less than the current buffer length.

public int compareTo (Object aCompareString)

This method compares the current string to aCompareString.

Object aCompareString is the string you want to compare the current string to.

This method returns one of the following values:

Value	Meaning
0	The strings are lexicographically equivalent.
< 0	The current string is lexicographically less than <code>aCompareString</code> .
> 0	The current string is lexicographically greater than <code>aCompareString</code> .

aCompareString Values

public void convert (UniSession aSession, Object aChars, Object aReplacementChars) throws UniStringException

public void convert (Object aChars, Object aReplacementChars) throws UniStringException

This method converts specified characters in the string buffer with other specified characters.

`UniSession aSession` is the `UniSession` object that establishes which server performs the operation.

`Object aChars` is the string of characters to be replaced.

`Object aReplacementChars` is the string of characters to replace `aChars`.

Every occurrence of a character specified by `aChars` is replaced by the corresponding character specified by `aReplacementChars`.

public int count()

public int count (Object aSubString)

The `count ()` method returns the number of field marks in the current string. The `count (Object aSubString)` method returns the number of occurrences of `aSubString` in the current string.

`Object aSubString` is the substring whose occurrences you want to count.

public int dcount()

public int dcount (Object aSubString)

The `dcount ()` method returns the number of field marks in the current string, plus 1. The `dcount (Object aSubString)` method returns the number of occurrences of `aSubString` in the current string, plus 1.

`Object aSubString` is the substring whose occurrences you want to count.

The `dcount ()` method differs from the `count ()` method in that it returns the number of values separated by delimiters rather than the number of occurrences of a character string.

public boolean equals (Object aCompareString)

This method determines whether this `UniString` is equivalent to another `Object`. It returns `true` only if the two objects represent the same sequence of characters.

`Object aCompareString` is the `Object` to compare the current string to.

This method overrides the `equals ()` method in the class `Object`.

public boolean equalsIgnoreCase (Object aCompareString)

This method determines whether this `UniString` is equivalent to another `Object` when case is ignored. It returns `true` only if the two objects represent the same sequence of characters.

`Object aCompareString` is the `Object` to compare the current string to.

This method overrides the `equals ()` method in the class `Object`.

public byte[] getBytes()

This method returns the current string as a `byte[]` array.

public void getChars (int srcBegin, int srcEnd, char dst[], int dstBegin) throws UniStringException

This method copies characters from the string buffer to a destination character array (dst).

int srcBegin is the first character to copy.

int srcEnd is the last character to copy.

char dst[] is the destination character array.

int dstBegin is the character in the destination array where the copied string should begin.

The total number of characters to copy is:

```
srcEnd - srcBegin
```

The characters are copied into the destination character array starting at aDstBegin and ending at:

```
dstBegin + (srcEnd - srcBegin) - 1
```

public String getMarkCharacter (int aTokenVal) throws UniStringException

This method returns the specified system delimiter. aTokenVal is one of the following:

Value	Token	Description
0	UniObjectsTokens.IM	Item mark
1	UniObjectsTokens.FM	Field mark
2	UniObjectsTokens.VM	Value mark
3	UniObjectsTokens.SVM	Subvalue mark
4	UniObjectsTokens.TM	Text mark

aTokenVal Values

***public UniString iconv (UniSession aSession, Object aConvCode)
throws UniStringException***

This method converts the contents of the string buffer to internal storage format using the specified conversion code.

UniSession aSession is the UniSession object representing which server to use to perform the conversion.

Object aConvCode is any BASIC ICONV conversion.

public void insert (int offset, Object object)

public void insert (int offset, char str[])

public void insert (int offset, boolean b)

public void insert (int offset, char c)

public void insert (int offset, int i)

public void insert (int offset, long l)

public void insert (int offset, float f)

public void insert (int offset, double d)

This method inserts a string into the string buffer.

int offset is the index of the character where the insertion begins. It must be a number greater than or equal to 0, and less than the current length of the string buffer.

Object object is an object whose value is converted to a string.

char str [] is a character string.

boolean b is a Boolean value, converted to a string.

`int i` is the number of characters to append.

`char c` is a character value.

`int i` is an integer value, converted to a string.

`long l` is a long numeric value, converted to a string.

`float f` is a floating decimal value, converted to a string.

`double d` is a double numeric value, converted to a string.

public UniString left (int aNumChars)

This method extracts a substring from the string buffer, starting from the leftmost character.

`int aNumChars` is the number of characters to extract.

public int length()

This method returns the length, or character count, of the string.

public UniString oconv (UniSession aSession, Object aConvCode) throws UniStringException

This method converts the contents of the string buffer to external format using the specified conversion code.

`UniSession aSession` is the `UniSession` object representing which server to use to perform the conversion.

`Object aConvCode` is any BASIC OCONV conversion.

public void quote()

public void quote (Object aChar)

The `quote()` method encloses the contents of the string buffer in double quotation marks. The `quote (Object aChar)` delimits the contents of the string buffer in a pair of characters specified by `aChar`.

Object `aChar` is the character to use as a delimiter.

public UniString right (int aNumChars)

This method extracts a substring from the string buffer, starting from the rightmost character.

`int aNumChars` is the number of characters to extract.

**public void setCharAt (int index, char c) throws
UniStringException**

This method sets the character at `index` to `c`.

`int index` is the index of the character to change. It must be a number greater than or equal to 0, and less than the current length of the string buffer.

`char c` is the new character you are setting.

public void setValue (Object newValue)

This method sets the current values of the string buffer and its size.

public int status ()

This method returns the status of the last operation.

public UniString substring (int aBegin) throws UniStringException

This method returns a substring of the current `UniString`, starting at the index `aBegin`.

`int aBegin` is where to begin the substring extraction.

***public UniString substring (int aBegin, int aEnd) throws
UniStringException***

This method returns a substring of the current `UniString`, starting at the index `aBegin` and ending at the endpoint `aEnd`.

int aBegin is where to begin the substring extraction.

int aEnd is where to end the substring extraction.

public char[] toCharArray()

This method returns the current string as a char [] array.

public UniString toLowerCase()

This method changes all the characters in the current string to lowercase letters.

public UniString toUpperCase()

This method changes all the characters in the current string to uppercase letters.

public String toString()

This method converts the contents of the string buffer to a string. It allocates and initializes a new `String` object to contain the characters in the current string buffer. Subsequent changes to the string buffer do not affect the contents of the `String` object.

UniStringTokenizer Object

The `UniStringTokenizer` object is a virtual extension of the `java.util.StringTokenizer` class with one major difference. Using the base `java` model, delimiters are treated as white space, and a series of delimiters is treated as a single delimiter. The database, however, needs to treat each system delimiter as separating a new token. The `nextToken` method reflects that difference.

UniStringTokenizer Object Constructors

public UniStringTokenizer (Object stringVal)
UniStringTokenizer (Object stringVal, Object delimiterVal)

UniStringTokenizer Object Methods

These are the methods you can use with the `UniStringTokenizer` object:

- `countTokens ()`
- `hasMoreTokens ()`
- `nextToken ()`
- `resetTokenizer ()`

public int countTokens ()

This method counts the number of tokens in a string. It returns an integer representing the number of tokens available in the string.

public boolean hasMoreTokens ()

This method determines if any tokens remain in the string. It returns a Boolean value representing whether any tokens remain.

public String nextToken ()

This method extracts the next string (token) that is separated by the delimiter. It returns `null` if there are no more tokens.

public void resetTokenizer()

This method resets the string tokenizer to look at the beginning of the string, allowing the string to be reparsed.

UniDynArray Object

The `UniDynArray` object lets you manipulate fields, values, and subvalues in a dynamic array such as a database record or a select list. `UniDynArray` objects are used in:

- Data in records of the `UniFile` and `UniDictionary` objects
- The `readList` method of the `UniSelectList` object

The `UniDynArray` object is a specific instance of the `UniString` object: it handles database strings that contain field marks, value marks, and subvalue marks. The `UniDynArray` object inherits the methods of the `UniString` object but overrides them with versions that work specifically on dynamic arrays.

The `UniDynArray` class converts an input string into a series of subobjects, each of which is inserted into a `Java Vector` object. Because of this, the dynamic array needs to be parsed only once, and `Vector` operations can easily manipulate the `UniDynArray` object.

For more information about the `UniDynArray` object, see [Fields, Values, and Subvalues](#) in Chapter 2, “Using UniObjects for Java.”

UniDynArray Object Constructors

public UniDynArray()

This is the default constructor for the class. It constructs a dynamic array with no characters in it. If a `UniSession` object instantiates it, the `UniDynArray` object inherits the system delimiters defined for that session; otherwise, it uses the standard default system delimiters.

UniDynArray (Object aString)

This constructs a dynamic array containing the value of `aString`.

Object `aString` is the data to be converted to a dynamic array.

UniDynArray Object Methods

These are the methods you can use with the UniDynArray object:

- `count()`
- `dcount()`
- `delete()`
- `extract()`
- `insert()`
- `length()`
- `remove()`
- `replace()`
- `toString()`

public int count()

public int count (int aField)

public int count (int aField, int aValue)

public int count (int aField, int aValue, int aSubValue)

This method counts one of the following:

- The number of field marks in the UniDynArray object
- The number of value marks in a field of the UniDynArray object
- The number of subvalue marks in a value of the UniDynArray object
- The number of text marks in a subvalue of the UniDynArray object

`int aField` is the field whose value marks, subvalue marks, or text marks you want to count.

`int aValue` is the value whose subvalue marks or text marks you want to count.

`int aSubValue` is the subvalue whose text marks you want to count.

This method overrides the `count()` method in the class `UniString`. It corresponds to the UniObjects **Count** method and the BASIC COUNT function.

public int dcount()

public int dcount (int aField)

public int dcount (int aField, int aValue)

public int dcount (int aField, int aValue, int aSubValue)

This method counts one of the following:

- The number of fields in the UniDynArray object
- The number of values in a field of the UniDynArray object
- The number of subvalues in a value of the UniDynArray object
- The number text values in a subvalue of the UniDynArray object

`int aField` is the field whose values, subvalues, or text values you want to count.

`int aValue` is the value whose subvalues or text values you want to count.

`int aSubValue` is the subvalue whose text values you want to count.

This method is equivalent to `count () + 1`. This method overrides the `dcount ()` method in the class `UniString`. It corresponds to the UniObjects **Count** method and the BASIC DCOUNT function.

public void delete (int aField)

public void delete (int aField, int aValue)

public void delete (int aField, int aValue, int aSubValue)

This method deletes the specified field, value, or subvalue from a dynamic array.

`int aField` is the number of the field you want to delete, or the number of the field containing the value or subvalue you want to delete.

`int aValue` is the number of the value you want to delete, or the number of the value containing the subvalue you want to delete.

`int aSubValue` is the number of the subvalue you want to delete.

This method corresponds to the UniObjects **Del** method and the BASIC DELETE function.

public UniDynArray extract()

public UniDynArray extract (int aField)

public UniDynArray extract (int aField, int aValue)

public UniDynArray extract (int aField, int aValue, int aSubValue)

This method extracts one of the following:

- The entire UniDynArray object as a dynamic array
- A field from the UniDynArray object
- A value from a field in the UniDynArray object
- A subvalue from a value in the UniDynArray object

`int aField` is the number of the field to extract, or the number of the field containing the value or subvalue to extract.

`int aValue` is the number of the value to extract, or the number of the value containing the subvalue to extract.

`int aSubValue` is the number of the subvalue to extract.

It corresponds to the BASIC EXTRACT function.

public void insert (int aField, Object aString)

public void insert (int aField, int aValue, Object aString)

public void insert (int aField, int aValue, int aSubValue, Object aString)

public void insert (int aField, int aValue, int aSubValue, Object aString)

This method inserts an object into a dynamic array, moving subsequent fields, values, or subvalues down.

`int aField` is the field to insert, or the number of the field into which to insert the value or subvalue.

`int aValue` is the number of the value to insert, or the number of the value into which to insert the subvalue.

`int aSubValue` is the number of the subvalue to insert.

`Object aString` is the string representing the data to insert.

This method corresponds to the UniObjects **Ins** method and the BASIC INSERT function.

public int length()

public int length (int aField)

public int length (int aField, int aValue)

public int length (int aField, int aValue, int aSubValue)

This method gets the length of the dynamic array, or of the specified field, value, or subvalue.

`int aField` is the number of the field whose length you want, or the number of the field containing the value or subvalue whose length you want.

`int aValue` is the number of the value whose length you want, or the number of the value containing the subvalue whose length you want.

`int aSubValue` is the number of the subvalue whose length you want.

This method corresponds to the UniObjects **Length** method.

public UniDynArray remove (int aField)

public UniDynArray remove (int aField, int aValue)

public UniDynArray remove (int aField, int aValue, int aSubValue)

This method deletes a field, value, or subvalue from the UniDynArray object, returning the field, value, or subvalue as a new UniDynArray object.

`int aField` is the number of the field to remove, or the number of the field containing the value or subvalue to remove.

`int aValue` is the number of the value to remove, or the number of the value containing the subvalue to remove.

`int aSubValue` is the number of the subvalue to remove.

public void replace (int aField, Object aString)

public void replace (int aField, int aValue, Object aString)

public void replace (int aField, int aValue, int aSubValue, Object aString)

This method replaces a field, value, or subvalue with a new field, value, or subvalue.

`int aField` is the number of the field whose value you want to replace, or the number of the field containing the value or subvalue you want to replace.

`int aValue` is the number of the value you want to replace, or the number of the value containing the subvalue you want to replace.

`int aSubValue` is the number of the subvalue you want to replace.

`Object aString` is the replacement string value.

This method corresponds to the UniObjects **Replace** method and the BASIC REPLACE function.

public String toString()

This method returns a string version of the dynamic array. It overrides the `toString()` method in the class `UniString`. It corresponds to the UniObjects **StringValue** property.

Example Using the UniDynArray Object

```
UniString custRec= orderFile.read( "SMITH,J" );
UniDynArray custArray = new UniDynArray( custRec );

System.out.println( "Customer Name = " + custRec.extract(
NAME.FIELD );

custRec.replace( STATE.FIELD, "IOWA" );
uFile.write( "SMITH,J" custRec );
```

UniRecord Object

The `UniRecord` object controls database record interaction. It is an extension of the `UniDynArray` object and behaves in a similar way. All `UniDynArray` operations can be performed on the data portion of the `UniRecord` object.

UniRecord Object Constructors

public UniRecord()

This constructs a `UniRecord` with no data in it.

public UniRecord (Object aRecID)

This constructs a `UniRecord` comprising only the record ID.

`aRecID` is the record ID.

public UniRecord (Object aRecID, Object aRecVal)

This constructs a `UniRecord` comprising the record ID and the record's data.

`aRecID` is the record ID.

`aRecVal` is the record's data.

public UniRecord (Object aRecID, Object aRecVal, int aStatus)

This constructs a `UniRecord` comprising the record ID and the record's data, and setting the initial status.

`aRecID` is the record ID.

`aRecVal` is the record's data.

`aStatus` is the initial status value of the record.

UniRecord Object Methods

These are the methods you can use with the UniRecord object:

- `getRecord()`
- `getRecordID()`
- `returnCode()`
- `setRecord()`
- `setRecordID()`
- `setReturnCode()`
- `setStatus()`
- `status()`
- `toString()`
- `toUniDynArray()`
- `toUniString`

public UniDynArray getRecordID()

This method returns the UniRecord object's record ID as a UniDynArray object.

public UniDynArray getRecord()

This method returns the UniRecord object's data value as a UniDynArray object.

public int returnCode()

This method returns an integer representing the UniRecord object's return code.

public int status()

This method returns an integer representing the UniRecord object's status. This method overrides `status` in the class `UniString`.

public void setRecordID (Object aRecID)

This method sets the UniRecord object's record ID.

`aRecID` is a UniDynArray object representing the record ID you want to set.

public void setRecord (Object aRecVal)

This method sets the UniRecord object's data value.

aRecVal is a UniDynArray object representing the UniRecord object's data value.

public void setReturnCode (int aReturnCode)

This method sets the UniRecord object's return code.

aReturnCode is an integer representing the return code.

public void setStatus (int aStatusVal)

This method sets the UniRecord object's status.

aStatusVal is an integer representing the status.

public String toString()

This method returns the UniRecord object as a String object. The record ID and the record's data value are combined, separated by an item mark. This method overrides toString in the class UniDynArray.

public UniDynArray toUniDynArray()

This method returns the UniRecord object as a UniDynArray object.

public UniString toUniString()

This method returns the UniRecord object as a UniString object.

UniDataSet Object

The `UniDataSet` object is a collection object. It provides a collection interface for sets of `UniRecord` objects, which can then be used to perform bulk or batch operations with one network operation.

UniDataSet Object Constructors

public UniDataSet()

public UniDataSet (Object initVal)

`initVal` is a string representing the initial record IDs to store in the data set. The record IDs are separated by the `UniObjectsTokens.AT_FM` character.

public UniDataSet (Object initVal, String delimiter)

`initVal` is a string representing the initial record IDs to store in the data set. The record IDs are separated by the `UniObjectsTokens.AT_FM` character.

`delimiter` is a string representing the delimiter to use to separate the record IDs.

UniDataSet Object Methods

These are the methods you can use with the `UniDataSet` object:

- absolute()
- afterLast()
- append()
- close()
- deleteRow()
- findRow()
- first()
- getCurrentRow()
- getDataSet()
- getIDSet()
- getString()
- getUniDynArray()
- getUniRecord()
- getUniString()
- getRowCount()
- insert()
- isAfterLast()
- isBeforeFirst()
- isFirst()
- isLast()
- last()
- next()
- previous()
- relative()
- setIndex()
- toString()

public boolean absolute (int rowNum)

This method specifies the absolute position in the UniDataSet that the cursor should point to. It returns a Boolean value indicating whether the operation was successful.

rowNum is an integer specifying the absolute position.

public void afterLast()

This method moves the internal UniDataSet cursor to point to the end of the data set.

public boolean append (Object rowID)

public boolean append (Object rowID, Object rowData)

public boolean append (UniRecord recordSet)

This method appends a new data element to the end of the existing data set. It returns a Boolean value indicating whether the operation was successful.

rowID is an object identifying the added data.

rowData is an object specifying the data to add.

`recordSet` is a `UniRecord` object specifying the data to add.

public void close()

This method closes the data set, resetting internal values to their initial state.

public boolean deleteRow()

public boolean deleteRow (int indexLoc)

public boolean deleteRow (String recordID)

This method deletes the current row from the data set. It returns a Boolean value indicating whether the operation was successful.

`indexLoc` is an integer representing the row to delete.

`recordID` is a string representing the record ID of the record to delete.

public int findRow (String aRowVal)

This method returns an integer representing the cursor position of a row based on the record ID passed in. If the record ID is found, the index value is passed back; if it is not found, `-1` is returned.

`aRowVal` is the name of the field to reference.

public void first()

This method moves the internal `UniDataSet` cursor to point to the beginning of the data set.

public int getCurrentRow()

This method returns an integer representing the current cursor position in the data set.

public String getDataSet()

This method returns the data contained in the data set as a `String` of data elements separated by item marks.

public String getIDSet()

This method returns the record IDs contained in the data set as a `String` of record IDs separated by item marks.

public String getString ()

public String getString (int columnIndex)

public String getString (string columnName)

This method returns the data set row represented by `currentRow` as a `String`.

`columnIndex` is an integer representing the data row to retrieve.

`columnName` is an integer representing the data row to retrieve.

public UniDynArray getUniDynArray()

public UniDynArray getUniDynArray (int indexLoc)

public UniDynArray getUniDynArray (String columnName)

This method extracts the row at the current cursor position, returning it as a `UniDynArray` object.

`indexLoc` is an integer representing the row at the current cursor position.

`columnName` is a string representing the record ID of the row to return.

public UniRecord getUniRecord()

public UniRecord getUniRecord (int indexLoc)

public UniRecord getUniRecord (String columnName)

This method extracts the row at the current cursor position, returning it as a UniRecord object.

indexLoc is an integer representing the row at the current cursor position.

columnName is a string representing the record ID of the row to return.

public UniString getUniString()

public UniString getUniString (int indexLoc)

public UniString getUniString (String columnName)

This method extracts the row at the current cursor position, returning it as a UniString object.

indexLoc is an integer representing the row at the current cursor position.

columnName is a string representing the record ID of the row to return.

public int getRowCount()

This method returns an integer representing the size of the data set.

public boolean insert (Object rowID)

public boolean insert (int indexLoc, Object rowID)

public boolean insert (Object rowID, Object rowVal)

public boolean insert (int indexLoc, Object rowID, Object rowVal)

public boolean insert (UniRecord recordSet)

public boolean insert (int indexLoc, UniRecord recordSet)

This method inserts a new row into the data set at the current cursor position. It returns a Boolean value indicating whether the operation was successful.

rowID is the record ID of the row to insert.

indexLoc specifies where to insert the row.

rowVal is the data value of the row to insert.

recordSet is the UniRecord object representing the entire row to insert.

public boolean isAfterLast ()

This method returns a Boolean value indicating whether the cursor is positioned past the last row in the data set. Use this method to determine when the list is exhausted.

public boolean isBeforeFirst ()

This method returns a Boolean value indicating whether the cursor is positioned before the first row in the data set.

public boolean isFirst ()

This method returns a Boolean value indicating whether the cursor is positioned at the first row in the data set.

public boolean isLast()

This method returns a Boolean value indicating whether the cursor is positioned at the last row in the data set.

public void last()

This method sets the cursor at the last row in the data set.

public boolean next()

This method increments the data set cursor by one. It returns `true` if the cursor position can be moved. It returns `false` if the cursor is already at the end of the data set.

public boolean previous()

This method decrements the data set cursor by one. It returns `true` if the cursor position can be moved. It returns `false` if the cursor is already at the beginning of the data set.

public boolean relative (int numRows)

This method positions the data set cursor to a position `numRows` away from the current position. For example, if the cursor is already set to the third row and `UniDataSet.relative(5)` is referenced, the cursor is set to the eighth position in the data set. This method returns `true` if the operation succeeds. If the operation tries to move the cursor past the end or before the beginning of the data set, it returns `false` and sets the cursor to the last row or first row, respectively.

`numRows` is an integer representing the number of rows to move the cursor.

public boolean setIndex (int indexLoc)

This method sets the cursor position to `indexLoc`. It returns `true` if the operation is successful; it returns `false` if the operation tries to position the cursor outside the data set.

`indexLoc` is an integer representing the index location to use for the data set.

public String toString ()

This method converts the data set to a `String`, adding item marks to separate each row of the data set. This method overrides the `toString` method of the class `Object`.

UniSelectList Object

The `UniSelectList` object lets you manipulate a select list on the server. Select lists are described in [The Database Environment](#) and [Select Lists](#) in Chapter 2, “Using UniObjects for Java.”

UniSelectList

This is the default constructor for this class. Do not instantiate `UniSelectList` directly; instead, create it through the `UniSession.selectList()` method.

UniSelectList Object Methods

These are the methods you can use with the `UniSelectList` object:

- `clearList()`
- `formList()`
- `getEncryptionType()`
- `getList()`
- `isLastRecordRead()`
- `next()`
- `readList()`
- `saveList()`
- `select()`
- `selectAlternateKey()`
- `selectMatchingAK()`
- `setEncryptionType()`

public void clearList() throws UniSelectListException

This method clears a select list. It corresponds to the UniObjects **ClearList** method and the BASIC CLEARSELECT statement.

```
uSel.clearList();  
uSel.select( uFile );
```

public void formList (Object aString) throws UniSelectListException

public void formList (UniDataSet aSet) throws UniSelectListException

This method creates a select list from a supplied list of record IDs or a UniDataSet object.

Object `aString` is either a UniDynArray object or a delimited string containing a list of record IDs. If `aString` is a delimited String, the record IDs must be separated by field marks (`UniTokens.AT_FM`).

UniDataSet `aSet` is a UniDataSet object.

This method corresponds to the UniObjects **FormList** method and the BASIC FORMLIST statement.

```
UniString newSel = new UniString("111"+UniTokens.AT_FM+"222");
uSel.formList( newSel );
```

public int getEncryptionType()

This method returns the current state of this object's encryption setting, which is one of the following:

Value	Token	Description
0	UniObjectsTokens.NO_ENCRYPT	Do not encrypt data. This is the default value.
1	UniObjectsTokens.UV_ENCRYPT	Encrypt data using internal database encryption.

getEncryptionType Values

public void getList (Object aListName) throws UniSelectListException

This method activates the named list from the `&SAVEDLISTS&` file on the server.

Object `aListName` is the name of the list you want to activate.

This method corresponds to the UniObjects **GetList** method and the database GET.LIST command.

public boolean isLastRecordRead()

This method returns `true` if you try to read beyond the end of a select list, or if the select list is empty. Otherwise it returns `false`.

public UniString next() throws UniSelectListException

This method returns the next record ID in the select list. If the list is exhausted, `next()` returns `null`, and the `isLastRecordRead()` method returns `true`.

For example:

```
while (!uSelect.isLastRecordRead ())
{
    rec = uFile.read (uSelect.next ());
    <...process record...>
}
```

This method corresponds to the UniObjects **Next** method and the BASIC READNEXT statement.

public UniDynArray readList() throws UniSelectListException

This method gets the entire contents of a select list. It corresponds to the UniObjects **ReadList** method and the BASIC READLIST statement.

```
UniFile uFile = uSession.open( "CUST.FILE" );
UniSelect uSel = uSession.selectList ( 1 );
uSel.select( uFile );
UniDynArray custList = uSel.readList();
/*... process dynamic array */
```

public void saveList (Object aListName) throws UniSelectListException

This method saves the currently active select list in the `&SAVEDLISTS&` file.

Object `aListName` is the name of the list you save.

This method corresponds to the UniObjects **SaveList** method and the SAVE.LIST command.

public void select (UniFile uFile) throws UniSelectListException

***public void select (UniDictionary uvd) throws
UniSelectListException***

This method creates a select list containing all record IDs from the database file represented by the UniFile or UniDictionary object.

UniFile uFile is the UniFile object created by the UniSession.open () method.

UniDictionary uvd is the UniDictionary object created by the UniSession.openDict () method.

The new select list overwrites any previous select list and resets the select list pointer to the first record in the list.

This example opens the ORDERS file, creates a select list of its record IDs, then starts to read records from the file using the select list:

```
UniDataFile uFile = uSession.open ("ORDERS");  
UniSelectList uSelect = uSession.selectList (0);  
UniRecord uvr;  
uSelect.select (uFile);  
uvr = uFile.read (uSelect.next ());
```

This method corresponds to the UniObjects **Select** method, the BASIC SELECT statement, and the database SELECT command.

Note: The select () method does not correspond to the SQL SELECT statement.

***public void selectAlternateKey (UniFile uFile, Object aIndexName)
throws UniSelectListException***

***public void selectAlternateKey (UniDictionary uvd, Object
aIndexName) throws UniSelectListException***

This method creates a select list from values in the specified secondary index.

UniFile uFile is the UniFile object created by the UniSession.open () method.



UniDictionary uvd is the UniDictionary object created by the UniSession.openDict () method.

aIndexName is the name of a secondary index as specified in a database CREATE.INDEX command.

If the named secondary index does not exist, the select list is empty. The new select list overwrites any previous select list and resets the select list pointer to the first record in the list.

This method corresponds to the UniObjects **SelectAlternateKey** method and the BASIC SELECTINDEX statement.

```
uSel.selectAlternateKey( custFile, "CUST.ORDER.NO" );
```

public void selectMatchingAK (UniFile uFile, Object aIndexName, Object aIndexVal) throws UniSelectListException

public void selectMatchingAK (UniDictionary uvd, Object aIndexName, Object aIndexVal) throws UniSelectListException

This method creates a select list from the record IDs whose value matches that in a named secondary index field. The select list contains record IDs.

UniFile uFile is the UniFile object created by the UniSession.open () method.

UniDictionary uvd is the UniDictionary object created by the UniSession.openDict () method.

aIndexName is the name of a secondary index as specified in a database CREATE.INDEX command. If the index you specify does not exist, an empty select list is returned and the lastRecordRead () method returns true.

aIndexVal is a value from the secondary index. Records are selected when aIndexVal matches the value of the indexed field. It is equivalent to the following database SELECT command:

```
SELECT filename WITH indexname = indexvalue
```

The new select list overwrites any previous select list and resets the select list pointer to the first record in the list.

This method corresponds to the UniObjects **SelectMatchingAk** method and the BASIC SELECTINDEX statement.

```
uSel.selectMatchingAK( custFile, "CUST.STATE", "MA" );
```

public int setEncryptionType (int aEncryptType)

This method sets up the default encryption for all UniSelectList object operations.

int aEncryptType is one of the following:

Value	Token	Description
0	UniObjectsTokens.NO_ENCRYPT	Do not encrypt data (default).
1	UniObjectsTokens.UV_ENCRYPT	Encrypt data using internal database encryption.

aEncryptType Values

If an encryption type is set, all data transferred between client and server is encrypted.

Example Using the UniSelectList Object

```
UniDataFile uFile = uSession.open ("DATAFILE");
UniSelectList uSelect = uSession.selectList (0);
UniRecord uvr;
uSelect.select (uFile);
uvr = uFile.read (uSelect.next ());
while (!uSelect.isLastRecordRead ())
{
    uvr = uFile.read (uSelect.next ());
    <...process record...>
}
}
catch (UniSelectListException)
{
    /*deal with exception */
}
}
```

UniCommand Object

The `UniCommand` object manages the running of a database command on the server. The `UniSession` object controls execution of the command.

You can run only one command at a time during a session. For more information about using database commands, see [Using Database Commands](#) in Chapter 2, “Using UniObjects for Java.”

UniCommand()

This is the constructor for the class. Do not instantiate the `UniCommand` object as a stand-alone object; instead, create it through the `UniSession.command()` method.

UniCommand Object Methods

These are the methods you can use with the `UniCommand` object:

- `cancel()`
- `exec()`
- `getAtSelected()`
- `getBlockSize()`
- `getCommand()`
- `getEncryptionType()`
- `getSystemReturnCode()`
- `nextBlock()`
- `reply()`
- `response()`
- `setBlockSize()`
- `setCommand()`
- `setEncryptionType()`
- `status()`

public void cancel() throws UniCommandException

This method cancels all outstanding output from the executing command. It can be called only when the command status returned by the `status()` method is either `UVS_REPLY` or `UVS_MORE`. If the `cancel()` method is successful, the command status is reset to `UVS_COMPLETE`, allowing another command to be executed.

This method corresponds to the UniObjects **Cancel** method.

public void exec() throws UniCommandException

This method executes the command set up by the `setCommand()` method.

Use the `response()` method to get the results from executing the command. If an error occurs, `exec()` throws a `UniCommandException` and the `response()` method returns the error message produced by the executed command.

The `status()` method gets the current status of the command, that is, whether it has completed or is waiting for further input.

This example executes the command `LIST VOC SAMPLE 10` on the server:

```
UniCommand uvc = uSession.command ();
uvc.setCommand ("LIST VOC SAMPLE 10");
uvc.exec ();
```

This method corresponds to the UniObjects **Exec** method and the BASIC EXECUTE statement.

public int getAtSelected()

This method returns the value of the `@SELECTED` variable when the command has finished successfully. It corresponds to the UniObjects **AtSelected** property.

public int getBlockSize()

This method returns the size, in bytes, of the buffer used to hold the contents of the command response. The initial value is 0, which means no limit to the size of the buffer and all data should be returned.

public String getCommand()

This method returns the current execution string set by the `setCommand()` method.

public int getEncryptionType()

This method returns the default encryption for all `UniCommand` object operations, which is one of the following:

Value	Token	Description
0	<code>UniObjectsTokens.NO_ENCRYPT</code>	Do not encrypt data (default).
1	<code>UniObjectsTokens.UV_ENCRYPT</code>	Encrypt data using internal database encryption.

getEncryptionType Values

If an encryption type is set, all data transferred between client and server is encrypted.

public int getSystemReturnCode()

This method gets the value of the `@SYSTEM.RETURN.CODE` returned by the command on the server. It corresponds to the `UniObjects` **SystemReturnCode** property.

public void nextBlock() throws UniCommandException

This method gets the next block of data from the command response if the command response generates more data than fits in one block. Call the `response()` method to get the new block of data. Call the `status()` method to determine the current command status.

This method corresponds to the `UniObjects` **NextBlock** method.

public void reply (Object aReplyString) throws UniCommandException

This method replies to a command execution. Many commands require a user response. Use the `reply()` method to issue the correct response to a command. Call this method whenever the `status()` method returns `UVS_REPLY`.

Object `aReplyString` is the string to send to the server as a response.

This method corresponds to the UniObjects **Reply** method.

```
UniCommand runCmd = uFile.command( "RUN BP FOO" ) ;

runCmd.exec();
if ( runCmd.status() == UniObjectsTokens.UVS_REPLY )
{
    /* Command may need to respond to a 'Press y to continue' */
    runCmd.reply( "Y" );
}
```

public String response()

This method returns the output from the `exec()` and `reply()` methods. This is the output generated by the command on the server.

This method corresponds to the UniObjects **Response** property.

public void setBlockSize (int aBlockSize)

This method determines the size, in bytes, of the buffer used to hold the contents of the `response()` method. The initial value is 0, which means no limit to the size of the buffer.

If you expect a command to generate large quantities of data, you can set the block size to a manageable value and read the output in blocks. You read successive blocks with the `nextBlock()` method. In this case the `status()` method returns `UVS_MORE` when the buffer is full, and when you call the `response()` method, the next block of command output is read from the server.

***Note:** In a client/server application, running server commands that produce large quantities of output can decrease performance and increase network traffic. For more information about this, see [Client/Server Design Considerations](#) in Chapter 2, “Using UniObjects for Java.”*

This method corresponds to the UniObjects **BlockSize** property.

public void setCommand (String aCommandString)

This method specifies the command string to execute on the server. It corresponds to the UniObjects **Text** property.



aCommandString is the command to run on the server.

This example sets up a database command for execution:

```
uvc.setCommand ("LIST VOC SAMPLE 10");
```

public void setEncryptionType (int aEncryptType)

This method sets up the default encryption for all UniCommand object operations.

int aEncryptType is one of the following:

Value	Token	Description
0	UniObjectsTokens.NO_ENCRYPT	Do not encrypt data. This is the default value.
1	UniObjectsTokens.UV_ENCRYPT	Encrypt data using internal database encryption.

aEncryptType Values

If an encryption type is set, all data transferred between client and server is encrypted.

public int status()

This method returns the current command status. The status is one of the following:

Value	Token	Description
0	UniObjectsTokens.UVS_COMPLETE	The command finished execution or was cancelled. A new command can be executed.
1	UniObjectsTokens.UVS_REPLY	The server is waiting for a reply. This reply can be sent through the <code>reply()</code> method.
2	UniObjectsTokens.UVS_MORE	More data is waiting to be retrieved. This occurs if the command response generates more data than fits in one block.

status Values

If you use the `setBlockSize()` method to set the block size to a value other than 0, the `response()` method returns a data segment equivalent to the size that is set. If the command results are more than can fit in one block, call the `nextBlock()` method until `status()` returns `UVS_COMPLETE`.

This method corresponds to the UniObjects **CommandStatus** property.

Example Using the UniCommand Object

```
import asjava.uniobjects.*;
public static void main (String args[])
{
    UniJava uJava = new UniJava ();
    UniSession uSession = uJava.openSession ();
    try{
        uSession.connect ("alfa", "davem", "password", "uv");
        UniCommand uvc = uSession.command ();
        uvc.setCommand ("LIST VOC SAMPLE 10");
        uvc.exec ();
        System.out.println (uvc.getResponse ());
    }
    catch (UniCommandException)
    {
    }
}
```

UniSubroutine Object

The `UniSubroutine` object lets you run a cataloged BASIC subroutine on the server. For information about subroutines, see [Client/Server Design Considerations](#) in Chapter 2, “Using UniObjects for Java.”

UniSubroutine()

This is the constructor for the object. Do not instantiate the `UniSubroutine` object as a stand-alone object; instead, access it through the `UniSession.subroutine()` method.

UniSubroutine Object Methods

These are the methods you can use with the `UniSubroutine` object:

- `call()`
- `getArg()`
- `getEncryptionType()`
- `getNumArgs()`
- `getRoutineName()`
- `resetArgs()`
- `setArg()`
- `setEncryptionType()`
- `setNumArgs()`
- `setRoutineName()`

public void call() throws UniSubroutineException

This method executes the cataloged BASIC subroutine identified by the `setRoutineName()` method, or identified during the `UniSession.subroutine()` call.

For example:

```
UniSubroutine uSub = uSession.subroutine("SAMPLESUBR", 3);
uSub.setArg(0, "David");
uSub.setArg(1, "Thomas");
uSub.setArg(2, "Meeks");
uSub.call();
```

This method corresponds to the UniObjects **Call** method and the BASIC CALL statement.

public String getArg (int aArgNum) throws UniSubroutineException

This method retrieves argument values returned from the subroutine following a successful subroutine `call()`.

`int aArgNum` is the number of the argument you are requesting. The first argument is 0.

This method corresponds to the UniObjects **GetArg** method.

public void getEncryptionType()

This method returns the default encryption for all `UniSubroutine` object operations, which is one of the following:

Value	Token	Description
0	UniObjectsTokens.NO_ENCRYPT	Do not encrypt data. This is the default value.
1	UniObjectsTokens.UV_ENCRYPT	Encrypt data using internal database encryption.

getEncryptionType Values

If an encryption type is set, all data transferred between client and server is encrypted.

public int getNumArgs()

This method returns the number of arguments this subroutine expects to use.

public String getRoutineName()

This method returns the name of the subroutine to call on the server. It corresponds to the UniObjects **RoutineName** property.

public void resetArgs()

This method resets all arguments of the UniSubroutine object to their initial values. It corresponds to the UniObjects **ResetArgs** method.

***public void setArg (int aArgNum, Object aArgVal) throws
UniSubroutineException***

This method sets the value of an argument for a subroutine.

`int aArgNum` is the number of the argument you are setting. The first argument is 0.

`Object aArgVal` is the value of the argument to pass to the server subroutine. The argument is passed to the server before making the call. Any argument you do not specify with the `setArg ()` method is passed as an empty string.

For example:

```
UniSubroutine uSub = uSession.subroutine("SAMPLESUBR", 3);  
uSub.setArg(0, "David");  
uSub.setArg(1, "Thomas");  
uSub.setArg(2, "Meeks");
```

This method corresponds to the UniObjects **SetArg** method.

public void setEncryptionType (int aEncryptType)

This method sets up the default encryption for all UniSubroutine object operations.

`int aEncryptType` is one of the following:

Value	Token	Description
0	<code>UniObjectsTokens.NO_ENCRYPT</code>	Do not encrypt data. This is the default value.
1	<code>UniObjectsTokens.UV_ENCRYPT</code>	Encrypt data using internal database encryption.

aEncryptType Values

If an encryption type is set, all data transferred between client and server is encrypted.

public void setNumArgs (int aNumArgs) throws UniSubroutineException

This method sets the number of arguments to use with this subroutine.

`int aNumArgs` is the number of arguments.

public void setRoutineName (Object aRoutineName)

This method sets the name of the subroutine to call.

`Object aRoutineName` is the name of the cataloged subroutine.

This method corresponds to the `UniObjects RoutineName` property.

Example Using the UniSubroutine Object

```
UniSubroutine uSub = uSession.subroutine("SAMPLESUBR",
    3);
uSub.setArg(0, "David");
uSub.setArg(1, "Thomas");
uSub.setArg(2, "Meeks");
//uSub.setEncryptionType(1);

System.out.println("Subroutine set up, routine name = " +
    uSub.getRoutineName());
System.out.println("Subroutine - EncryptionType being
used
    = " + uSub.getEncryptionType());
System.out.println("Subroutine: Arg0 = " +
uSub.getArg(0));
System.out.println("                Arg1 = " +
```

```

uSub.getArg(1));
    System.out.println("          Arg2 = " +
uSub.getArg(2));

        System.out.println("Calling subroutine...");
        uSub.call();
        System.out.println("Subroutine finished... ");
        System.out.println("Subroutine: Arg0 = " +
uSub.getArg(0));
        System.out.println("          Arg1 = " +
uSub.getArg(1));
        System.out.println("          Arg2 = " +
uSub.getArg(2));

        System.out.println("Results displayed, resetting
args...");
        uSub.resetArgs();
        System.out.println("Subroutine: Arg0 = " +
uSub.getArg(0));
        System.out.println("          Arg1 = " +
uSub.getArg(1));
        System.out.println("          Arg2 = " +
uSub.getArg(2));l

```

UniTransaction Object

The `UniTransaction` object is available from the `UniSession` object. The `UniTransaction` object provides methods to start, commit, and roll back transactions for a session. If a session closes while transactions are active, the server rolls them back. For any `UniSession` object, only one transaction can be active at a time.

UniTransaction()

This is the default constructor for this class. Do not instantiate a `UniTransaction` object as a stand-alone object. Access it only through the `UniSession.transaction()` method.

UniTransaction Object Methods

These are the methods you can use with the `UniTransaction` object:

- `commit()`
- `getLevel()`
- `isActive()`
- `rollback()`
- `begin()`

public void commit() throws UniTransactionException

This method commits an active transaction. If it is a nested transaction, the parent transaction becomes active and the transaction level is decremented.

This method corresponds to the `UniObjects` **Commit** method and the BASIC **COMMIT** statement.

public int getLevel() throws UniTransactionException

This method returns the current transaction level. It corresponds to the `UniObjects` **Level** property.

Note: This method applied only to UniVerse.



public void isActive() throws UniTransactionException

This method determines if a transaction is active. It returns `true` if the transaction is active, otherwise it returns `false`. A transaction is currently active if the `UniTransaction.start()` method has been called, but neither `UniTransaction.commit()` nor `UniTransaction.rollback()` have been called.

This method corresponds to the UniObjects **IsActive** method.

public void rollback() throws UniTransactionException

This method rolls back an active transaction. If this is a nested transaction, the parent transaction becomes active and the transaction level is decremented.

This method corresponds to the UniObjects **Rollback** method and the BASIC `ROLLBACK` statement.

public void begin() throws UniTransactionException

This method begins a new transaction. This transaction can be nested. If a transaction is already active, the nested transaction becomes active and the transaction level is incremented.

This method corresponds to the UniObjects **Start** method and the BASIC `BEGIN TRANSACTION` statement.

Example Using the UniTransaction Object

```
UniTransaction uvt = uSession.transaction();

/* Ok, let's open up a file and first write a record
outside
    & the transaction */
UniFile uFile = uSession.open("FOOBAR");
UniString uvstr = uFile.read("JAVA");
UniString uvnewstr = new UniString("This is a test of
    & Transactions 1 ");
uFile.write("TRANSREC", uvnewstr.getValue());

byte bArray[] = new byte[128];
System.out.print("Data written outside transaction...
check on
    & it ");
```

```

        System.in.read(bArray);

        System.out.println("Starting transaction");
        System.out.println("Current transLevel = " +
uvt.getLevel());
        System.out.println("Is it active? " + uvt.isActive());
        uvt.rollback();
        uvt.begin();
        System.out.println("Transaction started: Level " +
            å uvt.getLevel());

        uFile.write("TRANSCOMMITREC", uvnewstr.getValue());
        System.out.print("Data written, but not committed... hit
any
            å key to continue");
        System.in.read(bArray);
        if (bArray[ 0 ] == 'Y')
        {
            uvt.commit();
            System.out.println("Committed task... hit any key to
            å continue");
            System.in.read(bArray);
        }
        else
        {
            uvt.rollback();
            System.out.println("Rolledback... hit any key to
continue");
            System.in.read(bArray);
        }
        System.out.println();
        System.out.println("Closing session");
        uvjava.closeSession(uSession);

    }
    catch (UniSessionException e)
    {
        System.out.println("UniSessionError: MessageID=" +
            å e.getErrorCode());
        System.out.println("UniSessionError: MessageText=" +
            å e.getMessage());
    }
    catch (UniFileException e)
    {
        System.out.println("DataSourceError");
    }
    catch (IOException e)
    {
        System.out.println("IO Error");
    }
}

```

UniNLSLocale Object (UniVerse Only)

The `UniNLSLocale` object applies only to UniVerse systems.

On UniVerse systems the `UniNLSLocale` object defines and manages the conventions in use. The five conventions are Time, Numeric, Monetary, Ctype, and Collate. The `UniNLSLocale` object allows these five names to be supplied as a single `UniDynArray` object, with five fields containing the relevant locale name. Locale names are derived from the client system and a defaultable locale identifier. The `UniNLSLocale` object is available from the `UniSession` object through the `UniSession.nlsLocale()` method. If NLS is disabled on the server, the `UniNLSLocale` object is not available, and `nlsLocale()` throws an exception.

NLSLocale()

This is the default constructor for the class. Do not instantiate it directly; instead, create it from the `UniSession.nlsLocale()` method.

UniNLSLocale Object Methods

These are the methods you can use with the `UniNLSLocale` object:

- `getClientNames()`
- `getServerNames()`
- `setName()`

public UniDynArray getClientNames()

This method returns a dynamic array of the locale names requested by the client. This is the locale specification as the client sees it.

This method corresponds to the UniObjects **ClientName** method.

public UniDynArray getServerNames() throws UniNLSException

This method returns the dynamic array of locale names as reported by the server. These can differ from the names returned by the `getClientNames()` method because of a mapping between client and server naming styles.

This method corresponds to the UniObjects **ServerName** method.

public void setName (Object aName) throws UniNLSException

public void setName (Object aName, int anIndex) throws UniNLSException

This method sets the NLS locale.

Object aName is either a UniString or a UniDynArray object containing either one element or five elements.

int anIndex specifies a particular locale category to set. anIndex is one of the following:

Index	Category	Token
1	Time	UniObjectsTokens.UVT-NLS_TIME
2	Numeric	UniObjectsTokens.UVT-NLS_NUMERIC
3	Monetary	UniObjectsTokens.UVT-NLS_MONETARY
4	Ctype	UniObjectsTokens.UVT-NLS_CTYPE
5	Collate	UniObjectsTokens.UVT-NLS_COLLATE

aIndex Values

If aName contains five elements, each locale category is set to the corresponding element in aName. If aName contains only one element, anIndex specifies the category to set. If int anIndex is omitted, all five categories are set to the value of aName.

When the name has been changed successfully, the getServerNames () and getClientNames () methods return the corresponding values.

This method corresponds to the UniObjects **SetName** method.

UniNLSMap Object (UniVerse Only)

The `UniNLSMap` object applies only to UniVerse systems.

The UniVerse server uses NLS maps to determine which map to use for a client's string data.

The `UniNLSMap` object is available from the `UniSession` object. If NLS is disabled on the server, the `UniNLSMap` object is not available, and `nlsMap()` throws an exception.

UniNLSMap()

This is the default constructor for the class. Do not instantiate it directly; instead, create it from the `UniSession.nlsMap()` method.

Note: Do not redefine NLS maps while an application has open sessions to a server.



UniNLSMap Object Methods

These are the methods you can use with the `UniNLSMap` object:

- `getClientNames()`
- `getMarkCharacter()`
- `getServerNames()`
- `setName()`

public String getClientNames()

This method returns the name of the map requested by the client. On the server it is mapped through the `NLS.CLIENT.MAPS` file to the name reported by the `getServerNames()` method.

This method corresponds to the UniObjects **ClientName** property.

public String getMarkCharacter (int aTokenVal) throws UniNLSException

This method returns the value of the specified system delimiter. Call this method, especially with an NLS-enabled server, to determine what are the proper values for each system delimiter.

int aTokenVal is the system delimiter you want. It is one of the following:

Value	Token	Description
1	UniObjectsTokens.IM	Item mark
2	UniObjectsTokens.FM	Field mark
3	UniObjectsTokens.VM	Value mark
4	UniObjectsTokens.SM	Subvalue mark
5	UniObjectsTokens.TM	Text mark
6	UniObjectsTokens.SQLNULL	Null value

getMarkCharacter Values

This method corresponds to the UniObjects **FM**, **IM**, **SQLNULL**, **SVM**, **TM**, and **VM** properties.

public String getServerNames () throws UniNLSException

This method returns the name of the map as reported by the server. This is the name that is loaded into the server shared memory segment.

This method corresponds to the UniObjects **ServerName** property.

public void setName (Object aMapName) throws UniNLSException

This method sets the map to use on the server.

Object aMapName is the name of the requested map.

When the name has been changed successfully, the getServerNames () and getClientNames () methods return the corresponding values.

This method corresponds to the UniObjects **SetName** method.

UniException Object

public class UniException extends Exception

This object is the error object thrown when a UniObjects for Java error or exception occurs. It is the error object caught during a try/catch loop. All applications can catch the generic `UniException` error object and determine the error number, type, and text. They can also catch specific `UniException` subclasses if they want more specific control over particular errors.

UniException()

UniException (String aMessageText, int aMessageNumber)

UniException (String aClassName, String aMessageText, int aMessageNumber)

This is the default constructor for the exception class. It establishes the error number of the error text passed in.

`String aMessageText` is the text of the error message associated with the exception.

`int aMessageNumber` is the error number assigned to the exception. See Appendix A, “[Error Codes and Replace Tokens](#),” for valid error codes for valid error codes.

`String aClassName` is the name of the class that threw the exception.

UniException Object Methods

These are the methods you can use with the `UniException` object:

- `getErrorCode()`
- `getMessage()`
- `getErrorMessage()`
- `getExtendedMessage()`

- `toString()`

public int getErrorCode()

This method returns the error code associated with the exception. See Appendix A, “[Error Codes and Replace Tokens](#),” for valid error codes.

public String getMessage()

This method returns the text of the error message.

public UniException getErrorType()

This method returns the exception type, which is one of the following:

Value	Token
0	UNIOBJECTS_EXCEPTION
1	UNISTRING_EXCEPTION
2	UNIDYNARRAY_EXCEPTION
3	UNICOMMAND_EXCEPTION
4	UNIFILE_EXCEPTION
5	UNINLS_EXCEPTION
6	UNISELECTLIST_EXCEPTION
7	UNISEQUENTIALFILE_EXCEPTION
8	UNISESSION_EXCEPTION
9	UNISUBROUTINE_EXCEPTION
10	UNITRANSACTION_EXCEPTION

UniException getErrorType Values

public String getExtendedMessage()

This method returns extended information, including the class or object and the method that threw the exception.

UniXML Object

The UniXML object represents an XML representation of UniData or UniVerse data. Using this class, you can create XML documents and XML Schema documents from UniQuery, UniData SQL, Retrieve, UniVerse SQL, or directly from a data file. UniData and UniVerse also provide functions to generate new data, modify data, or generate XML from the UniData or UniVerse database using the XMAP file.

The following table describes the private strings available with the UniXML class:

Private String	Description
private string m_Xsdstr	String type to store XML schema
private string m_Xmlstr	String type variable to store XML document
private string m_Errstr	Error message related to generated error code
private int m_Errorcode	Error Code

Private Strings for UniXML

Following are public methods used to get and set the private string variables:

- getXML
- setXML
- getXsd
- setXsd
- getErrmsg
- getErrcode

Public Methods

This section describes the public methods available with the UniXML class.

***public void generateXML(string command) throws
UniXMLException***

command is any UniQuery or UniVerse LIST or UniData or UniVerse SQL SELECT command not containing any XML-related keywords. UniObjects stores the resulting XML schema and XML document in the private variables *m_Xmlstr* and *m_Xsdstr*.

***public void generateXML(string command; string options) throws
UniXMLException***

This method executes a UniQuery, Retrieve, UniData SQL, or UniVerse SQL command, allowing you to pass options to XMLExecute(). You can separate each option by a space. The key and value pair can use “=” as well as @FM and @VM.

***public void generateXMLUsingXmap(string xmapname) throws
UniXMLException***

This method uses the XMAP residing on the server to generate an XML document and stores the result in *m_Xmlstr*.

***public void updateDataUsingXmap(string xmapname) throws
UniXMLException***

This method updates a file residing in the UniData or UniVerse database using the XMAP located in the *_XML_* directory in the UniData or UniVerse account. It uses the contents of the *m_Xmlstr* variable as the data.

***public void UpdateDataUsingXmap(string, xmapname, string
xmlname) throws UniXMLException***

This method updates a file residing in the UniData or UniVerse database using the XMAP located in the *_XML_* directory in the UniData or UniVerse account. It uses an XML document residing on the server as data.

Using SSL With UniObjects for Java

Overview of SSL Technology	5-3
Software Requirements	5-4
Setting up Java Secure Socket Extension (JSSE)	5-5
Configuring UOJ to use IBM JSSE	5-6
Configuring the Database Server for SSL	5-7
Creating a Secure Connection	5-9
Direct Connection	5-10
Establishing the Connection	5-12
Proxy Tunneling	5-13
Externally Secure	5-15
Managing Keys and Certificates for a UOJ Client and a Proxy Server	5-20
Importing CA Certificates Into UOJ Client Trustfile	5-20
Generating client certificates.	5-21
Managing Keyfile and Trustfile for the Proxy Server.	5-22

This chapter explains how to use SSL (Secure Socket Layer) with UniObjects for Java (UOJ). The topics covered include:

- “ Overview of SSL Technology”
- “ Software Requirements”
- “ Setting up Java Secure Socket Extension (JSSE)”
- “ Configuring UOJ to use IBM JSSE”
- “ Configuring the Database Server for SSL”
- “ Creating a Secure Connection”

Overview of SSL Technology

Secure Sockets Layer (SSL) is a transport layer protocol that provides a secure channel between two communicating programs over which arbitrary application data can be sent securely. It is by far the most widely deployed security protocol used on the World Wide Web.

Although it is most widely used in applications to secure web traffic, SSL actually is a general protocol suitable for securing a wide variety of other network traffic that is based on TCP, such as FTP and Telnet.

SSL provides server authentication, encryption and message integrity. It optionally also supports client authentication.

This document assumes that users who want to use this facility have some basic knowledge of public key cryptography.

For more information on the implementation of SSL with UniData and UniVerse, refer to *Developing UniBasic Applications* manual for UniData and the *Guide to UniVerse Basic* for UniVerse.

Software Requirements

You must have the following applications installed and configured on the client machine.

- JDK (Java Development Kit) 1.4 or higher
- UniObjects for Java version 2.0.0 or higher

Setting up Java Secure Socket Extension (JSSE)

The java.sun.com web site defines JSSE as a set of Java packages that enable secure Internet communications. JSSE implements a Java version of Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols and includes functionality for data encryption, server authentication, message integrity, and optional client authentication. Using JSSE, developers can provide for the secure passage of data between a client and a server running any application protocol (such as HTTP, Telnet, NNTP, and FTP) over TCP/IP.

SSL for UOJ requires an implementation of JSSE to be installed on the client computer as well as the proxy server if one is to be used.

UniObjects for Java ships with the IBM Reference implementation of JSSE, but any implementation from a valid JSSE provider should work. The file that contains the JSSE components is named `ibmjsse.jar` and is located in the archive directory of your UniDK installation, for example: `C:\IBM\UniDK\uojsdk\lib`.

Configuring UOJ to use IBM JSSE

First, copy `ibmjssse.jar` into the `/lib/ext` directory of your jdk installation or simply edit your `CLASSPATH` environment variable to reference the `ibmjssse.jar` file in the UOJ archive directory specified above.

Second, you will need to add the IBM JSSE provider to the list of security providers in the `java.security` file. This file is located in the jdk installation directory under `/lib/security`. Edit this file with Notepad or another text editor and add the following line:

```
security.provider.N=com.ibm.jsse.JSSEProvider
```

Where `N` is the number defining the position of the IBM JSSE in the list of security providers. For example, the file would look something like this.

```
security.provider.1=sun.security.provider.Sun
```

```
security.provider.N=com.ibm.jsse.JSSEProvider
```



Note: *If you already have a JSSE security provider installed on the client machine, there is no need to install the IBM JSSE unless you specifically want to use it. If you do decide to use the IBM JSSE, we recommend that you remove any other JSSE security providers to avoid any conflicts or problems.*

Configuring the Database Server for SSL

First, you need to create a Server Security Context Record (SCR).

A SCR contains all SSL related properties necessary for the server to establish a secured connection with an SSL client. The properties include the server's private key, certificate, client authentication flag and strength, and trusted entities. For more information, see *UniBasic Extensions*.

The SCR can be generated by directly calling the UniData or UniVerse Security API from a BASIC program, or alternatively, by invoking UniAdmin.

The SCR is encrypted by a password and saved in a UniData or UniVerse security file with a unique ID. The path, password and ID of the SCR for a UOJ server are important in the following descriptions.

In order to enable SSL support for UOJ on the database server you need to edit two configuration files, *unirpcservices* and *.scrfile*. Both of these files are located in the unishared/unirpc directory. On UNIX systems, you can determine the location of the unishared directory by entering *cat /.unishared*. On Windows platforms, the default location can be found by examining the registry record at `HKEY_LOCAL_MACHINE\SOFTWARE\IBM\UniShared`.

First, on the database server, edit the unirpcservices file. Open the file with a text editor such as vi on UNIX or Notepad on Windows, and locate the line that corresponds to the UOJ server. The line is similar to the following example:

```
udcs C:\IBM\ud71\bin\udapi_server.exe *TCP/IP 0 3600
```

Append "SCR-ID password" to the end of this line as shown in the following example, where "SCR-ID" is the record ID of your Security Context Record.

```
udcs C:\IBM\ud71\bin\udapi_server.exe *TCP/IP 0 3600 SCR-ID password
```

Now, edit the *.scrfile*. Refer to the section above to determine its location. This file contains the path to the Security Context Record store, which contains the Security Context Record specified in the "unirpcservices" file. The file format is as follows:

```
service-name path
```

For example:

```
udcs c:\IBM\ud71\demo
```

Once these files have been edited appropriately, the database server should be properly configured.

Creating a Secure Connection

There are three different modes you can use to establish a secure SSL session with a UniData or UniVerse database server.

- **Direct Connection** - This method is completely secure. In this mode the SSL session is established directly between the UOJ client and the UniData or UniVerse database server.
- **Proxy Tunneling** - This method is completely secure. In this mode, the connection is created through a proxy server. The proxy server provides tunneling for the data exchange between the UOJ client and the UniData or UniVerse database server. Since the proxy server does not decrypt data packets, there is no session multiplexing performed.
- **Externally Secure Proxy** - The security of this method is reliant on the external proxy. In this mode, the externally secure SSL session is established between the UOJ client and an external proxy server. The connection between the proxy server and the UniData or UniVerse database server is not a secure connection. A typical application for this type of connection would be in the case where both the proxy server and UniData or UniVerse database server are behind a firewall. Thus, the unsecured connection between the proxy and database server does not compromise security. In this mode, session multiplexing can be achieved.

The first step is to create a UniSession object by calling the **openSession** method of the **UniJava** object. The signature of the method is shown in the following example.

```
public UniSession openSession(int sslmode) throws  
UniSessionException
```

The *sslmode* parameter can be one of the following values:

Mode	Option
Direct Connection	UniObjectTokens.SECURE_SESSION
Proxy Tunnel	UniObjectTokens.SECURE_SESSION
Secure External Proxy	UniObjectTokens.EXTERNALLY_SECURE_PROXY_SESSION

Next, determine which of the following connection types you wish to use for the secure connection.

Direct Connection

When creating a secure connection, there are three components that you must consider. They are SSL Socket Factory, Cipher Suites and Keyfile, and Trustfile Parameters. You can define these parameters by creating and setting properties of the **UniSSLDescriptor** object associated with the secure session and setting some system variables.

- **SSL Socket Factory** - Secure Socket Factories encapsulate details for creating and initially configuring secure socket connections. The **SSLSocketFactory** object is a concrete implementation of the abstract **SocketFactory** class provided with JSSE in the `javax.net` package. It acts as a factory for creating secure sockets. You can define your own **SSLSocketFactory** object with the **setSSLSocketFactory** method of the **UniSSLDescriptor** object. If you pass a null parameter to this method, the system defaults will be used. Another way to use the system defaults is to set the **UniSSLDescriptor** object to null by calling the **setSSLDescriptor** method of the **UniSession** object with a null parameter.
- **Cipher Suites** - Define your own available Cipher Suites with the **setEnabledCipherSuites** method of **UniSSLDescriptor**. If you pass a null parameter to this method, the system defaults will be used.
- **Keyfile and Trustfile Parameters** - System Variables must be created to define locations of the keyfile, trustfile and the password to access these files. This step is required for any secure connection.

If **uniojbects.UniSSLDescriptor** is set to null, the system will use the system defaults for **SSLSocketFactory** and default Cipher suites.

Once you have created the session object, to specify your own **SSLSocketFactory** object and/or define available cipher suites, you need to create the **uniojbects.UniSSLDescriptor** using the constructor with the following signature.

```
public UniSSLDescriptor (void)
```

Once created, you need to call the **setSSLSocketFactory** method to set the SSL Socket Factory and **setEnabledCipherSuites** to set the available cipher suites and then pass this object to the session.

Calling the **setSSLSocketFactory** method with the signature shown in the following example will set **SSLSocketFactory**.

```
public void setSSLSocketFactory(SSLSocketFactory sslsf)
```

Calling the **setEnabledCipherSuites** method with the signature shown in the following example sets CipherSuites.

```
public void setEnabledCipherSuites(String [] cs)
```

Whether you specify your own Socket Factory and Cipher Suites or use the system defaults, you still need to specify the system variables for the location and password for the keyfile and the trustfile as shown in the following table:

System Variable	Definition
javax.net.sslTrustStore	Defines the location of the trustfile.
javax.net.sslKeyStore	Defines the location of the keyfile.
javax.net.sslKeyStorePassword	Defines the password for the keyfile.

The trustfile (also called truststore), is a file that holds a set of keys and certificates. In fact, the keyfile (also called keystore) has exactly the same format. The difference between a trustfile and a keyfile is more a matter of function than of a programming construct. The keyfile provides credentials for the secure connection and the trustfile verifies those credentials. The trustfile and keyfile can be, and often are, the same file.

You can use tools such as IBM's ikeyman utility and Sun's keytool to create and maintain the keyfile and trustfile. The Keytool utility is installed with Sun Microsystem's JDK. For more information on keytool, see <http://java.sun.com/products/jdk1.2/docs/tooldocs>. The default location for the trustfile (truststore) is \$JREHOME/lib/security/jssecacerts. If the file does not exist, the system assumes that the trustfile is located under \$JREHOME/lib/security/cacerts. There is no default location for the keyfile (keystore).

Establishing the Connection

Once you have set the secure parameters for the session, you can connect by calling the connect method of the **UniSession** object as you would in any normal, nonsecure session.

The following code example demonstrates how to create a secure Direct Connection with the database server.

Database server Connection Properties:

```
U2 host: localhost
user name:"test"
password:"new.pass"
accountpath: "demo"
```

Security Properties are:

```
keyfile path: "testkeys"
keyfile password: "new.pass"
trustfile path: "testtrust"
trustfile password: "new.pass"
```

```
String U2host = "localhost";
String username = "test";
String password = "new.pass";
String accountpath = "demo";
String keyfilepath = "testkeys";
String keyfilepwd = "new.pass";
String trusfilepath = "testkeys";

// First, let's instantiate our new UOJ application
uvJava = new UniJava();

// Now, let's open up a session
UniSession demoSession =
uvJava.openSession(UniObjectsTokens.SECURE_SESSION);

demoSession.setHostPort(UniRPCTokens.UNIRPC_DEFAULT_PORT );
demoSession.setHostName(U2host );
demoSession.setUserName( username );
demoSession.setPassword( password );
demoSession.setAccountPath( accountpath );

// Now we'll set locations for the keystore and truststore and a password
for the keystore

System.setProperty("javax.net.sslTrustStore", "testtrust");
System.setProperty("javax.net.sslKeyStore", "testkeys");
System.setProperty("javax.net.sslKeyStorePassword.", "new.pass");

demoSession.setSSLDescriptor(null);
demoSession.connect();
```

Proxy Tunneling

The process for using the Proxy Tunneling method is basically the same as the Direct Connection method. The only difference is that the connection is tunnelled through a proxy server which passes messages between the client and database server. There are no additional parameters to configure but the proxy server should be properly configured, as described in Chapter 3, “[Using the Proxy Server.](#)”

You need to set the `PROXY_SSL_FLAG` parameter in the `uniproxy.config` file to true, so the proxy server will listen for secure connections. See “[Externally Secure](#)” on page 5-14 for more information on editing the `uniproxy.config` file.

The following example demonstrates how to create a secure connection with the database server through a Proxy Tunneling server.

The U2 connection properties are:

```
U2 host: localhost
user name:"test"
password:"new.pass"
accountpath: "demo"
```

Proxy server properties are:

```
Proxy host - localhost
Proxy token - "password1"
```

Security properties are:

```
keyfile path: "testkeys"
keyfile password:
trustfile path: "testkeys"
"new.pass"
```

```

String U2host = "localhost";
String username = "test";

String password = "new.pass";
String accountpath = "demo";

String proxyhost = "localhost";
String proxytoken = "password1";

String keyfilepath = "testkeys";
String keyfilepwd = "new.pass";
String trusfilepath = "testkeys";

int sslmode = UniObjectsTokens.SECURE_SESSION;

// Instantiate our new Uni/Java application
UniJava uvJava = new UniJava();

// First, let's open up a session
UniSession demoSession = uvJava.openSession(sslmode);
demoSession.setHostName( U2Host );
demoSession.setHostPort(UniRPCTokens.UNIRPC_DEFAULT_PORT );
demoSession.setUserName(username );
demoSession.setPassword( password );
demoSession.setAccountPath( accountPath );
demoSession.setProxyHost(proxyhost);
demoSession.setProxyPort(UniRPCTokens.UNIRPC_DEFAULT_PROXY_PORT);
demoSession.setProxyToken(proxytoken);

// Set system variables for locations of the keystore and
truststore and a password for the keystore

System.setProperty("javax.net.sslTrustStore", "testtrust");
System.setProperty("javax.net.sslKeyStore", "testkeys");
System.setProperty("javax.net.sslKeyStorePassword.", "new.pass");

// use default SSLSocketFactory object
demoSession.setSSLDescriptor(null);
demoSession.connect();

```

Externally Secure

This method requires that you define the properties described in the *uniproxy.config* file. For more information on editing the *uniproxy.config* file see [“Editing the Proxy Server Configuration File,”](#) in Chapter 3, [“Using the Proxy Server.”](#)

You must set the following parameters for SSL for UOJ configuration.

Parameter	Description
PROXY_SSL_FLAG	This parameter enables or disables externally secure connections. Its value can be <i>true</i> or <i>false</i> . When set to <i>true</i> , the proxy server will start a new thread that listens on PROXY_SSL_PORT for externally secure connections. This parameter must be set to <i>true</i> for both Proxy Tunneling and Externally Secure modes. The default setting is <i>false</i> .
PROXY_SSL_ONLY_FLAG	If this parameter is set to <i>true</i> , the proxy only allows secure connections to pass through to the database server. The default setting is <i>false</i> .
PROXY_SSL_PORT	This parameter defines the port on which the proxy server should listen for externally secure connections.
SSL_KEY_FILE	This parameter specifies the location of the keyfile (keystore).
SSL_TRUST_FILE	This parameter specifies the location of the trustfile (truststore).
SSL_KEY_FILE_TYPE	This parameter specifies the type of the proxy server keyfile type. It can be either <i>JKS</i> or <i>JCEKS</i> . The default value is <i>JKS</i> .
SSL_TRUST_FILE_TYPE	This parameter specifies the type of the proxy server trustfile. It can be either <i>JKS</i> or <i>JCEKS</i> . The default value is <i>JKS</i> .

Parameter	Description
SSL_PWD_METHOD	<p>This parameter defines the method in which password for the keystore is specified. This parameter can take the following values:</p> <p><i>DIRECT</i> - When this value is selected, the password is stored directly in the SSL_KEY_FILE_PWD.</p> <p><i>USER_DEFINED</i> - When you select this value, the parameter, SSL_KEY_FILE_PWD contains a description of how to call a user defined java method that will generate the password. In this case, the value for these properties consists of three fields separated by the underscore character, “_”. The first field is a parameter for the method and should be of type String. The second field is a method name and a third field defines a class name. This mode provides better security for protecting the passwords. However, keep in mind that it may be possible that the password algorithm can be reverse engineered.</p> <p><i>INTERACTIVE</i> - When you select this value, the proxy server prompts the user to enter a password for the keyfile and trustfile interactively during the startup. This mode provides the most password security but cannot support proxy auto-restart.</p>
SSL_KEY_FILE_PWD	This parameter contains information depending on settings defined in the SSL_PWD_METHOD.
SSL_CLIENT_AUTHENTICATION	This parameter specifies whether or not the proxy will ask for a client certificate during the SSL handshake.

The following example demonstrates how to create an Externally Secure connection with the database server.

- The keyfile (keystore) that contains credentials (keys and certificate) for the proxy server is called "testkeys" and is located in the current proxy directory.
- The keyfile type is JKS.
- The proxy server should authenticate all UOJ clients.
- The trustfile (truststore) that contains trusted certificates is called "testtrust" and is located in the current proxy directory.

- The trustfile type is JKS.
- The passwords for the keystore and truststore should be entered interactively.
- The proxy port for listening for externally secure connections is 31452.

The proxy configuration for this example is as follows:

```
PROXY_SSL_FLAG=true
PROXY_SSL_PORT=31452
SSL_KEY_FILE=testkeys
SSL_TRUST_FILE=testtrust
SSL_KEY_FILE_TYPE=JKS
SSL_TRUST_FILE_TYPE=JKS
SSL_PWD_METHOD=INTERACTIVE
SSL_CLIENT_AUTHENTICATION=true
```

database server: localhost

```
user name:newuser
password:new.pass
accountpath: demo
```

Proxy server properties are:

```
Proxy host: localhost
Proxy token: password1
```

Security properties are:

```
keyfile path: testkeys
keyfile password: new.pass
trustfile path: testtrust

String U2host = localhost;
String username = newuser;

String password = new.pass;
String accountpath = demo;

String proxyhost = localhost;
String proxytoken = password1;

String keyfilepath = testkeys;
String keyfilepwd = new.pass;
String trusfilepath = testkeys;;

int sslmode = UniObjectsTokens.EXTERNALLY_SECURE_PROXY_SESSION;

// Instantiate our new Uni/Java application
UniJava uvJava = new UniJava();

// First, let's open up a sessions
UniSession demoSession = uvJava.openSession(sslmode);

    demoSession.setHostName( U2Host );

demoSession.setHostPort(UniRPCTokens.UNIRPC_DEFAULT_PORT );

demoSession.setUserName(username );
demoSession.setPassword( password );
demoSession.setAccountPath( accountPath );

    demoSession.setProxyHost(proxyhost);

demoSession.setProxyPort(UniRPCTokens.UNIRPC_DEFAULT_SSL_PROXY_PORT);

    demoSession.setProxyToken(proxytoken);

// Set locations for the keystore and truststore and a password for the
keystore
System.setProperty(javax.net.sslTrustStore, testtrust);
System.setProperty(javax.net.sslKeyStore, testkeys);
System.setProperty(javax.net.sslKeyStorePassword, new.pass);

// use default SSLSocketFactory object
    demoSession.setSSLDescriptor(null);

demoSession.connect();
```

Managing Keys and Certificates for a UOJ Client and a Proxy Server

When a server establishes a secure session with a client, it passes its certificate down for authentication. The client usually has a list of trusted certificates that it uses to verify server credentials. If the client cannot verify the server certificate through its trusted certificates, it rejects the connection. Optionally, a server may also require a client to authenticate itself by providing the server with a valid trusted certificate. In the case where the server cannot verify the client certificate, the secure connection is not established. A list of trusted certificates that is used to verify credentials usually resides in a trustfile, and private keys and certificates providing credentials are kept in the keyfile.

A UOJ client should provide the system with a location of trustfile and keyfile and also the keyfile password by setting system properties.

The JDK usually contains a program that works with keyfiles and trustfiles. In Sun Microsystems's implementation of the JDK, this utility is called **keytool**. In IBM's JDK implementation it is called the **ikeman** utility. All examples from this chapter use the **keytool** utility. For a complete description of **keytool** utility, see ["http://java.sun.com/products/jdk/1.4/docs/toddocs/win32/keytool.html"](http://java.sun.com/products/jdk/1.4/docs/toddocs/win32/keytool.html).

Importing CA Certificates Into UOJ Client Trustfile

In general, a server's certificate is issued by a trusted third party called a Certificate Authority (CA), whose certificate (CA certificate) is used to sign the server certificate. In order for a client to verify a server's certificate, the UOJ client should import the trusted server's CA certificate into its trustfile.

Suppose we have a trusted server CA certificate in the file `cacert.pem`, the client's trustfile is called *testtrust*, and the access password for the trustfile is *passphrase*. By executing the following command, you can import the certificate into the trustfile.

```
keytool -import file cacert.pem -keystore testtrust -storepass  
passphrase
```

Generating client certificates

In the case where the database server or the proxy server requires client authentication, the client certificate should be generated and installed into the client's keyfile. Complete the following steps below to generate and install the certificate for the client.

1. Generate a key pair consisting of a public key and a private key. The following command in the **keytool** utility generates an RSA type key pair, as well as a self-signed certificate in the keyfile.

```
keytool -genkey -keystore testkeys -storepass passphrase -keyalg  
RSA
```

2. Create a certificate request. The following command in the **keytool** utility creates a certificate request in the file javacert.req.

```
keytool -certreq keystore testkeys -storepass passphrase -file  
javacert.req
```

3. Send a certificate request to a Certificate Authority (CA). The javacert.req file containing the certificate request should be sent to a valid Certificate Authority that will approve it and send back the certificate chain. We assume that the certificate chain is returned in the file javacert.pem file. A file javacert.pem can be exported to the client keyfile.

If you choose to use the UniData BASIC API to generate certificates for requests, or if the CA described in the previous paragraph returns its CA certificate separately, the server CA certificate should be separately installed into the client's keystore before generated certificates are installed there. The CA Certificate must be imported into the keyfile using an alias, as described in the following example.

```
keytool -import -file cacert.pem -keystore testkeys -storepass  
passphrase -alias ca
```

Where cacert.pem contains the CA certificate and ca is the name of the alias.

4. Replace your own certificate with the newly created CA-signed certificate in the keyfile. The following command in the **keytool** utility will replace the self-signed certificate with the newly generated one.

```
keytool -import -file javacert.pem -keystore testkeys -storepass  
passphrase
```

Managing Keyfile and Trustfile for the Proxy Server.

The keyfile and trustfile for the proxy server should be managed by a standard key and certificate utility, such as Sun Microsystem's **keytool** or IBM's **ikeman** utility.

Error Codes and Replace Tokens

UniObjects for Java provides replace tokens for error codes and global constants that may be useful in your application. They are contained in the file whose path is

C:\IBM\UniClient\UNIDK\INCLUDE\UVOAIF.TXT. You can add this file to an application through the **Add File** option of the File menu.



***Note:** UVOAIF.TXT is a generic file used by client programs accessing the database. This appendix describes only those tokens that are relevant to UniObjects for Java.*

Error Codes

These are the error codes that can be returned to a UniObjects for Java application, together with their replace tokens. Each token should be used with the `UniObjectsTokens` prefix—for example, `UniObjectsTokens.UVE_NOERROR`.

Code	Token	Description
0	UVE_NOERROR	No error
14002	UVE_ENOENT	No such file or directory
14005	UVE_EIO	I/O error
14009	UVE_EBADF	Bad file number
14012	UVE_ENOMEM	No memory available
14013	UVE_EACCES	Permission denied
14022	UVE_EINVAL	Invalid argument
14023	UVE_ENFILE	File table overflow
14024	UVE_EMFILE	Too many open files
14028	UVE_ENOSPC	No space left on device
14551	UVE_NETUNREACH	Network is unreachable
22001	UVE_BFN	Bad Field Number
22002	UVE_BTS	Buffer size too small
20003	UVE_IID	Illegal record ID
22004	UVE_LRR	The last record in the select list has been read
22005	UVE_NFI	Not a file identifier
30001	UVE_RNF	Record not found
30002	UVE_LCK	This file or record is locked by another user

Error Codes

Code	Token	Description
30095	UVE_FIFS	The file ID is incorrect for the current session
30097	UVE_SELFAIL	The select operation failed
30098	UVE_LOCKINVALID	The task lock number specified is invalid
30099	UVE_SEQOPENED	The file was opened for sequential access and you have attempted hashed access
30100	UVE_HASHOPENED	The file was opened for hashed access and you have attempted sequential access
30101	UVE_SEEKFAILED	The operation using <code>fileSeek()</code> failed
30103	UVE_INVALIDATKEY	The key used to set or retrieve an @variable is invalid
30105	UVE_UNABLETOLOADSUB	Unable to load the subroutine on the server
30106	UVE_BADNUMARGS	Too few or too many arguments supplied to the subroutine
30107	UVE_SUBERROR	The subroutine failed to complete successfully
30108	UVE_ITYPEFTC	The I-type operation failed to complete correctly
30109	UVE_ITYPEFAILEDTOLOAD	The I-type failed to load
30110	UVE_ITYPENOTCOMPILED	The I-type has not been compiled
30111	UVE_BADITYPE	This is not an I-type, or the I-type is corrupt
30112	UVE_INVALIDFILENAME	Must specify a filename
30113	UVE_WEOFFAILED	WEOFSEQ failed

Error Codes (Continued)

Code	Token	Description
30114	UVE_EXECUTEISACTIVE	An EXECUTE is currently active on the server
30115	UVE_EXECUTENOTACTIVE	No EXECUTE is currently active on the server
30124	UVE_TX_ACTIVE	Cannot perform this operation while a transaction is active
30125	UVE_CANT_ACCESS_PF	Cannot access part files
30126	UVE_FAIL_TO_CANCEL	Failed to cancel an execute
30127	UVE_INVALID_INFO_KEY	Bad key for the host type
30128	UVE_CREATE_FAILED	The creation of a sequential file failed
30129	UVE_DUPHANDLE_FAILED	Failed to duplicate a pipe handle
31000	UVE_NVR	No VOC record
31001	UVE_NPN	No pathname in VOC record
39101	UVE_NODATA	The server is not responding
39119	UVE_AT_INPUT	The server is waiting for input to a command
39120	UVE_SESSION_NOT_OPEN	The session is not open
39121	UVE_UVEXPIRED	The database license has expired
39122	UVE_CSVERSION	The client and the server are not running at the same release level
39123	UVE_COMMSVERSION	The client or server is not running at the same release level as the communications support
39124	UVE_BADSIG	You are trying to communicate with the wrong client or server
39125	UVE_BADDIR	The directory does not exist or is not a database account
39127	UVE_BAD_UVHOME	Cannot find the UV account directory

Error Codes (Continued)

Code	Token	Description
39128	UVE_INVALIDPATH	An invalid pathname was found in the UV.ACCOUNT file
39129	UVE_INVALIDACCOUNT	The account name supplied is not an account
39130	UVE_BAD_UVACCOUNT_FILE	The UV.ACCOUNT file could not be found or opened
39131	UVE_FTA_NEW_ACCOUNT	Failed to attach to the specified account
39134	UVE_ULR	The user limit has been reached on the server
39135	UVE_NO_NLS	NLS is not available
39136	UVE_MAP_NOT_FOUND	NLS map not found
39137	UVE_NO_LOCALE	NLS locale support not available
39138	UVE_LOCALE_NOT_FOUND	NLS locale not found
39139	UVE_CATEGORY_NOT_FOUND	NLS locale category not found
39201	UVE_SR SOCK_CON_FAIL	The server failed to connect to the socket
39210	UVE_SR_SELECT_FAIL	The server failed to select on input channel. When you see this error, you must quit and reopen the session.
39211	UVE_SR_SELECT_TIMEOUT	The select has timed out
40001	UVE_INVALIDFIELD	Pointer error in a sequential file operation
40002	UVE_SESSIONEXISTS	The session is already open
40003	UVE_BADPARAM	An invalid parameter was passed to a subroutine
40004	UVE_BADOBJECT	An incorrect object was passed
40005	UVE_NOMORE	The nextBlock () method was used but there are no more blocks to pass.

Error Codes (Continued)

Code	Token	Description
40006	UVE_NOTATINPUT	The <code>reply()</code> method can be used only when the <code>response()</code> method returns <code>UVS_REPLY</code>
40007	UVE_INVALID_DATAFIELD	The dictionary entry does not have a valid <code>TYPE</code> field
40008	UVE_BAD_DICTIONARY_ENTRY	The dictionary entry is invalid
40009	UVE_BAD_CONVERSION_DATA	Unable to convert the data in the field
45000	UVE_FILE_NOT_OPEN	File has been closed, must reopen before performing an operation
45001	UVE_OPENSESSION_ERR	Maximum number of UniJava sessions already open
45002	UVE_NONNULL_RECORDID	Cannot perform operation on a nonnull record ID
80011	UVE_BAD_LOGINNAME	The user name or login name provided is incorrect
80019	UVE_BAD_PASSWORD	The password has expired
80144	UVE_ACCOUNT_EXPIRED	The account has expired
80147	UVE_RUN_REMOTE_FAILED	Unable to run as the given user
80148	UVE_UPDATE_USER_FAILED	Unable to update user details
81001	UVE_RPC_BAD_CONNECTION	The connection is bad and may be passing corrupt data.
81002	UVE_RPC_NO_CONNECTION	The connection is broken
81005	UVE_RPC_WRONG_VERSION	The version of the UniRPC on the server is different from the version on the client.
81007	UVE_RPC_NO_MORE_CONNECTIONS	No more connections available

Error Codes (Continued)

Code	Token	Description
81009	UVE_RPC_FAILED	The UniRPC failed
81011	UVE_RPC_UNKNOWN_HOST	The host name specified is not valid, or the host is not responding
81014	UVE_RPC_CANT_FIND_SERVICE	Cannot find the service in the <i>unirpc-services</i> file
81015	UVE_RPC_TIMEOUT	The connection has timed out
81016	UVE_RPC_REFUSED	The connection was refused as the UniRPC daemon is not running
81017	UVE_RPC_SOCKET_INIT_FAILED	Failed to initialize the network interface
81018	UVE_RPC_SERVICE_PAUSED	The UniRPC service has been paused
81019	UVE_RPC_BAD_TRANSPORT	An invalid transport type has been used
81020	UVE_RPC_BAD_PIPE	Invalid pipe handle
81021	UVE_RPC_PIPE_WRITE_ERROR	Error writing to pipe
81022	UVE_RPC_PIPE_READ_ERROR	Error reading from pipe

Error Codes (Continued)

@ Variables

The following tokens represent BASIC @variables:

Value	Token	BASIC @Variable
1	AT_LOGNAME	@LOGNAME
2	AT_PATH	@PATH
3	AT_USERNO	@USERNO
4	AT_WHO	@WHO
5	AT_TRANSACTION	@TRANSACTION
6	AT_DATA_PENDING	@DATA.PENDING
7	AT_USER_RETURN_CODE	@USER.RETURN.CODE
8	AT_SYSTEM_RETURN_CODE	@SYSTEM.RETURN.CODE
9	AT_NULL_STR	@NULL.STR
10	AT_SCHEMA (UniVerse only)	@SCHEMA

BASIC @variables

Blocking Strategy Values

The following tokens set the blocking strategy:

Value	Token	Meaning
1	UVT_WAIT_LOCKED	If the record is locked, wait until it is released.
2	UVT_RETURN_LOCKED	Return a value to indicate the state of the lock. This is the default. The values that can be returned are shown in “Lock Status Values” on page A-12.

Command Status Values

The following tokens represent possible database command status values:

Value	Token	Meaning
0	UVS_COMPLETE	Execution of the command is complete.
1	UVS_REPLY	The command is waiting for a reply.
2	UVS_MORE	More output to come from the command; the command is waiting for a <code>nextBlock()</code> method.

Command Status Values

Host Type Values

The following tokens represent possible host type values:

Value	Token	Meaning
0	UVT_NONE	The host cannot be determined or is not yet connected.
1	UVT_UNIX	The host is a UNIX system.
2	UVT_NT	The host is a Windows system.

Host Type Values

Lock Status Values

The following tokens represent the values returned by the `status()` method to indicate the state of a lock:

Value	Token	Meaning
0	LOCK_NO_LOCK	The record is not locked.
1	LOCK_MY_READL	This user holds the READL lock.
2	LOCK_MY_READU	This user holds the READU lock.
3	LOCK_MY_FILELOCK	This user holds an exclusive file lock.
4	<i>no token</i>	This user holds a shared file lock.
-1	LOCK_OTHER_READL	Another user holds the READL lock.
-2	LOCK_OTHER_READU	Another user holds the READU lock.
-3	LOCK_OTHER_FILELOCK	Another user holds an exclusive file lock.
-4	<i>no token</i>	Another user holds a shared file lock.
PID	<i>no token</i>	Another user holds a shared file lock. The status value will be the process ID (PID) of the user holding the lock.

Lock Status Values

Locking Strategy Values

The following tokens set the locking strategy:

Value	Token	Meaning
0	UVT_NO_LOCKS	No locking. This is the default.
1	UVT_EXCLUSIVE_READ	Sets a READU lock.
2	UVT_SHARED_READ	Sets a READL lock.

Locking Strategy Values

fileSeek() Pointer Values

The following tokens indicate the relative position parameter values used with the `fileSeek()` method of the `UniSequentialFile` object:

Value	Token	Meaning
0	UVT_START	Start of file
1	UVT_CURR	Current position
2	UVT_END	End of file

fileSeek() Pointer Values

NLS Locale Values (UniVerse Only)

NLS locale values apply only to UniVerse systems. The following tokens represent the five NLS locale categories:

Value	Token	Category
1	UVT-NLS_TIME	Time
2	UVT-NLS_NUMERIC	Numeric
3	UVT-NLS_MONETARY	Monetary
4	UVT-NLS_CTYPE	Ctype
5	UVT-NLS_COLLATE	Collate

NLS Locale Values

Release Strategy Values

The following tokens set the release strategy:

Value	Token	Meaning
1	UVT_WRITE_RELEASE	Releases the lock when the record is written.
2	UVT_READ_RELEASE	Releases the lock when the record is read.
4	UVT_EXPLICIT_RELEASE	Maintains locks as specified by the lock strategy. Releases the locks only with the <code>unlockRecord()</code> method.
8	UVT_CHANGE_RELEASE	Releases the lock when a new record ID is set. This value is additive and can be combined with any of the other values.

Release Strategy Values

System Delimiters

The following tokens represent database system delimiters:

Value	Token	Character Value	Meaning
1	TM_CHAR	251	Text mark
2	SM_CHAR	252	Subvalue mark
3	VM_CHAR	253	Value mark
4	FM_CHAR	254	Field mark
5	IM_CHAR	255	Item mark

System Delimiters

Encryption Values

The following tokens set the encryption values:

Value	Token	Meaning
0	NO_ENCRYPTION	Do not encrypt data.
1	UV_ENCRYPT	Encrypt data using internal database encryption.

Encryption Values

The Demo Application

This appendix describes one of the demonstration applications that is supplied with UniObjects for Java for Java on your installation CD. The demonstration application uses some of the functionality provided by UniObjects for Java, displaying it graphically, showing you how UniObjects accesses UniVerse and UniData from the Java environment.

The appendix starts by describing where to find the demo and then explains the programming techniques used to build it.



***Note:** The demonstration program can be run either as an applet or as an application. We recommend that you run the demonstration program first as an application, as this is the simplest and most trouble-free way to run it.*



Installing and Running the Demo

All the files for the demonstration application can be found in the `uoj sdk\samples\demo` subdirectory of your UniDK installation directory.

Note: The database should be running on the server before you start.

Installing the Demonstration Program

To install the demonstration program, follow these steps:

1. Include the UniObjects for Java class libraries in your class path. For example:
`CLASSPATH=c:\unidk\uoj sdk\lib\asjava.zip`
2. Compile the demonstration application using an IDE development environment or Sun's Java SDK.
3. Run the compiled demonstration program as an applet with a Java interpreter. For example:

```
c:\unidk\uoj sdk\samples\demo\java FileDemo
```

Or put the web page and class files on a web server and use your web browser to run the application. For example:

```
http://www.webserver/filedemo/FileDemo.html
```

Or launch the class file through the applet viewer supplied with your Java development kit. For example:

```
c:\unidk\uoj sdk\samples\demo\appletviewer FileDemo.html
```

Running the Demonstration Program

When you run the demonstration program as an application or from the appletviewer, it can access any UniVerse or UniData server available on your network. If you run the demonstration applet in a web browser or some other restricted environment, only a database server on the web server is accessible.

To run the application from a web browser, we recommend that you put the demonstration applet classes and the *asjava.zip* package in the same directory on the web server, then access the applet through the web server. This will show you what users experience and what security issues are involved when the applet is accessed in this way.

To use the demonstration program, click each button on the left, in order. After you click a button, an action is performed and a detailed description of the classes and methods used to perform it are displayed.

When you click the **Connect** button, enter the logon information requested. Since the application supports environmental parameters, you can include this information in the *FileDemo.html* file.



***Note:** The **Exit Demo** button appears only when the demonstration program is run as an application. This is because an applet has no concept of closing its instance: the application that launched the applet controls its closing.*

Code Structure

The code for the demo application is held in two modules, as shown in the following table. These files are in the `uojsdk\samples\demo` directory. They are text files, so you can print them or inspect them using Notepad or another editor.

Category	Module Name	Description
Java	<i>FileDemo.java</i>	Java source code for the demonstration program.
HTML	<i>FileDemo.html</i>	HTML code to run the demonstration program from a web browser.

Modules Used in the Demo Application

`FileDemo` is a class constructed to show the use of some of the features of `UniObjects` for Java. This class has no other purpose and is not a product or representative of a product of IBM Corporation. This class uses the following `UniObjects` for Java classes:

- `UniSession`
- `UniFile`
- `UniSelectList`
- `UniCommand`
- `UniSubroutine`
- `UniDynArray`
- `UniString`

Program Initialization

```
import java.applet.*;
import java.awt.Event;
import java.awt.Frame;
import java.awt.event.*;
import java.awt.Label;
import java.awt.*;
import java.util.*;
import asjava.uniclientlibs.*;
import asjava.uniobjects.*;
```

Declaring and Initializing Variables

```
public class FileDemo extends Applet
{
    public boolean isApplet = true; // used for applet conditional
    code
    public Color buttonColor = new Color(0,191,255); // Deep Ski
    Blue 1
    public Color textColor = new Color(85,26,139); // Purple4
    private int lineCount = 0; // used for TextArea output
    public boolean cancelConnect = false;
        â // used for LogonDialog feedback
    public TextArea taScreen;
    private Button bConnect, bCreateFile, bLoadFile, bListFile,
        â bCreateSelectList, bShowSelectList, bExecuteCommand,
        â bBuildSubroutine, bExecuteSubroutine, bExitDemo,
        â bDisconnect;
    public UniSession session; // Database session
    public UniSelectList demoSelect;
        â // The select list for our demo file
    public Frame myFrame = null; // pointer to our current Frame
    public String userName, password, server, accountPath;
        â // user parameters
```

Main Entry Point

This is the main entry point if this class is launched as an application:

```
public static void main(String[] args)
{
    new FileDemoFrame( new FileDemo(), 620, 400 );
}
```

Initializing FileDemo

First the program reads the relevant parameters from the user's environment, which are all parameters set in the applet tag in your HTML file.

```
public void init()
{
    if (isApplet)
    {
        userName = getParameter("username");
        password = getParameter("password");
        server = getParameter("server");
        accountPath = getParameter("account");
    }
    else
    {
        userName = "";
        password = "";
        server = "";
        accountPath = "";
    }
}
```

Next the program adds all components to the applet:

```
setLayout( new BorderLayout());

Panel p = new Panel();

// only include an exit demo button if we are launching
// this program as an application.

if(isApplet)
{
    p.setLayout(new GridLayout(9,1));
}
else
{
    p.setLayout(new GridLayout(10,1));
}
bConnect = new Button("Connect UniVerse");
bConnect.setForeground(textColor);
bConnect.setBackground(buttonColor);
bCreateFile = new Button("Create File");
bCreateFile.setForeground(textColor);
bCreateFile.setBackground(buttonColor);
bLoadFile = new Button("Load File");
bLoadFile.setForeground(textColor);
bLoadFile.setBackground(buttonColor);
bListFile = new Button("List File");
bListFile.setForeground(textColor);
bListFile.setBackground(buttonColor);
bCreateSelectList = new Button("Create Select List");
bCreateSelectList.setForeground(textColor);
```

```

bCreateSelectList.setBackground(buttonColor);
bShowSelectList = new Button("Show Select List");
bShowSelectList.setForeground(textColor);
bShowSelectList.setBackground(buttonColor);
bExecuteCommand = new Button("Execute Command");
bExecuteCommand.setForeground(textColor);
bExecuteCommand.setBackground(buttonColor);
bExecuteSubroutine = new Button("Execute Subroutine");
bExecuteSubroutine.setForeground(textColor);
bExecuteSubroutine.setBackground(buttonColor);
bDisconnect = new Button("Disconnect UniVerse");
bDisconnect.setForeground(textColor);
bDisconnect.setBackground(buttonColor);

// only build an exit demo button if we are launching
// this program as an application.

if(!isApplet)
{
    bExitDemo = new Button("Exit Demo");
    bExitDemo.setForeground(textColor);
    bExitDemo.setBackground(buttonColor);
}

```

Next the program adds the buttons to the display panel:

```

p.add(bConnect);
p.add(bCreateFile);
p.add(bLoadFile);
p.add(bCreateSelectList);
p.add(bShowSelectList);
p.add(bListFile);
p.add(bExecuteCommand);
p.add(bExecuteSubroutine);
p.add(bDisconnect);

// only include an exit demo button if we are launching
// this program as an application.

if(!isApplet)
{
    p.add(bExitDemo);
}
add("West", p);
taScreen = new TextArea();
taScreen.setForeground(textColor);
add("Center", taScreen);

```

Next the program adds listeners for all buttons:

```
        bConnect.addActionListener( new ConnectListener( this ) );
        bCreateFile.addActionListener( new CreateFileListener( this
    ) );
        bLoadFile.addActionListener( new LoadFileListener( this ) );
        bCreateSelectList.addActionListener( new
            â CreateSelectListListener( this ) );
        bShowSelectList.addActionListener( new
            â ShowSelectListListener( this ) );
        bListFile.addActionListener( new ListFileListener( this ) );
        bExecuteCommand.addActionListener( new
            â ExecuteCommandListener( this ) );
        bExecuteSubroutine.addActionListener( new
            â ExecuteSubroutineListener( this ) );
        bDisconnect.addActionListener( new DisconnectListener( this
    ) );

        // only include an exit demo action listener if we are
        launching
        // this program as an application.

        if(!isApplet)
        {
            bExitDemo.addActionListener( new ExitDemoListener( this )
    );
        }
        this.resize(600, 400); // this works only if we are
            â an application
    }
```

Starting and Stopping FileDemo

```
public void start()
{
    // get a frame pointer to use for spawning the logon dialog.
    myFrame = getFrame(this);
}

public void stop()
{
    appOutput("Disconnecting from the Database Server.\n");
    if(session != null)
    {
        try
        {
            appOutput("  UniSession.disconnect()\n");
            session.disconnect();
        }
        catch (UniSessionException e)
    }
```

```

        {
            appOutput("    ERROR:UniSessionException:" +
e.getMessage()
                a + "\n");
        }
    }
}

```

Get Parent Window [Frame]

This section gets the parent frame. If this is an applet, this code is needed to get a parent window handle to launch a logon dialog box with.

```

static Frame getFrame(Component component)
{
    Frame frame = null;
    while((component = component.getParent()) != null)
    {
        if(component instanceof Frame)
            frame = (Frame)component;
    }
    return frame;
}

```

Controlling Data Output to Screen

This section controls the output of data to the screen.

```

public void appOutput(String output)
{
    // this is required because the peer for the text area
    control on
    // most platforms has a limited amount of text it can
    display at
    // one time. So we control that here.

    if( lineCount < 250)
    {

        // append this string to our TextArea

        taScreen.append(output);
        lineCount++;
    }
    else
    {

        // we have reached our maximum text threshold for the
        // TextArea. So we remove half the text and keep going
    }
}
until

```

```

        // we reach the threshold again .

        String tmp = taScreen.getText();
        int length = tmp.length();
        taScreen.setText(tmp.substring(length / 2));
        taScreen.append(output);
        lineCount = 126;
    }
}

```

Parent Frame

This section creates the parent frame if *FileDemo* is launched as an application. This frame runs the applet in the same way a web browser would.

```

class FileDemoFrame extends Frame
{
    FileDemo applet = null;
    FileDemoFrame(FileDemo childApplet, int x, int y)
    {
        applet = childApplet;
        setTitle("Ardent Java Demo");
        setSize(x, y);
        add("Center", applet);
        applet.isApplet = false;
        applet.init();
        setVisible(true);
        applet.start();

        // this listener detects all application exit signals

        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent event)
            {
                applet.stop();
                applet.destroy();
                dispose();
                System.exit(0);
            }
        });
    }
}

```

Accepting Logon Information for the Database

This section creates the dialog that accepts logon information for the database. The defaults can be specified as parameters if *FileDemo* is launched as an applet.

```
class LogonDialog extends Dialog
{
    // graphical elements for this dialog

    private Label lUserName, lPassword, lServer, lAccountPath;
    private TextField tUserName, tPassword, tServer, tAccount;
    private Button bOK, bCancel;
    private Color labelColor = new Color(139,0,0); // Dark Red
    private FileDemo parentApplet; // reference to our parent

    // construct frame

    LogonDialog(Frame f,FileDemo fileDemo) {
        this(f, fileDemo, "Database Logon");
    }

    // construct frame with a title

    LogonDialog(Frame f,FileDemo fileDemo,String title) {
        super(f,title,true);
        parentApplet = fileDemo;

        // initialize the layout manager

        GridBagLayout gbl = new GridBagLayout();
        setLayout(gbl);
        GridBagConstraints gbc1 = new GridBagConstraints();
        GridBagConstraints gbc2 = new GridBagConstraints();
        gbc1.weighty = 1.0;
        gbc2.weighty = 1.0;
        gbc1.anchor = GridBagConstraints.EAST;
        gbc2.anchor = GridBagConstraints.WEST;
        gbc2.gridwidth = GridBagConstraints.REMAINDER;

        // create the user text boxes

        tUserName = new TextField(parentApplet.userName,20);
        tUserName.setForeground(parentApplet.textColor);
        tPassword = new TextField(parentApplet.password,20);
        tPassword.setForeground(parentApplet.textColor);
        tPassword.setEchoChar('*');
        tServer = new TextField(parentApplet.server,20);
        tServer.setForeground(parentApplet.textColor);
        tAccount = new TextField(parentApplet.accountPath,20);
        tAccount.setForeground(parentApplet.textColor);

        // create the buttons
```

```

bOK = new Button("OK");
bOK.setForeground(parentApplet.textColor);
bOK.setBackground(parentApplet.buttonColor);
bCancel = new Button("Cancel");
bCancel.setForeground(parentApplet.textColor);
bCancel.setBackground(parentApplet.buttonColor);

// add all the components to the container

add(gbl, gbc1, lUserName=new Label("UserName:"));
lUserName.setForeground(labelColor);
add(gbl, gbc2, tUserName);
add(gbl, gbc1, lPassword=new Label("Password:"));
lPassword.setForeground(labelColor);
add(gbl, gbc2, tPassword);
add(gbl, gbc1, lServer=new Label("Server:"));
lServer.setForeground(labelColor);
add(gbl, gbc2, tServer);
add(gbl, gbc1, lAccountPath=new Label("Account Path:"));
lAccountPath.setForeground(labelColor);
add(gbl, gbc2, tAccount);
add(gbl, gbc1, bOK);
add(gbl, gbc2, bCancel);

// add action listeners for buttons and window closes

bOK.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        // pass the entered parameters to the parent applet

        parentApplet.userName = tUserName.getText();
        parentApplet.password = tPassword.getText();
        parentApplet.server = tServer.getText();
        parentApplet.accountPath = tAccount.getText();
        parentApplet.cancelConnect = false;
        setVisible(false);
        dispose();
    }
});
bCancel.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        // cancel connect operation

        parentApplet.cancelConnect = true;
        setVisible(false);
        dispose();
    }
});

```

```

    );
    addWindowListener(new WindowAdapter()
    {
        public void windowClosing(WindowEvent event)
        {
            setVisible(false);
            dispose();
        }
    }
    );

    Point scrnLoc = parentApplet.myFrame.getLocationOnScreen();
    setLocation(scrnLoc.x + 150, scrnLoc.y + 100);

    setSize(300,200);
    setVisible(true);

    /* for some reason requestFocus and transferFocus are not
    /* working */

    bOK.requestFocus();
    bOK.transferFocus();
}

// Macro for adding components

void add(GridBagLayout gb,GridBagConstraints c,Component o)
{
    gb.setConstraints(o,c);
    add(o);
}
}

```

Connecting to the Database

This section opens a connection to the database.

```

class ConnectListener implements ActionListener
{
    // handle to the base applet

    private FileDemo parentApplet;

    // constructor that requires a handle to the base applet

    ConnectListener(FileDemo handle)
    {
        parentApplet = handle;
    }

    // This method is executed whenever the Connect button is
    pressed.

```

```

public void actionPerformed(ActionEvent event)
{
    parentApplet.appOutput("Connecting to Database Server.\n");
    new LogonDialog(parentApplet.myFrame, parentApplet);

    // only connect if "OK" was pressed in logon dialog.

    if(!parentApplet.cancelConnect)
    {
        try
        {
            // create a new UniSession object

            parentApplet.session = new UniSession();

            // set the connect parameters to the session object

parentApplet.session.setUser_name(parentApplet.userName);

parentApplet.session.setPassword(parentApplet.password);
            parentApplet.session.setHostName(parentApplet.server);

parentApplet.session.setAccountPath(parentApplet.accountPath);
            parentApplet.appOutput("    UniSession.connect()
ã -- Hostname=" + parentApplet.session.getHostName() +
ã "=UserName=" + parentApplet.session.getUserName() +
ã "=AccountPath=" +
ã parentApplet.session.getAccountPath() + = "\n");

            // try to connect this session

            parentApplet.session.connect();
        }
        catch (UniSessionException e)
        {
            parentApplet.appOutput("    ERROR:UniSessionException:"
ã + e.getMessage() + "\n");
        }

        // check to see if our session connect succeeded

        if(parentApplet.session.isActive())
        {
            parentApplet.appOutput("Connection Successfully
ã established.\n");
        }
        else

```

```

        {
            parentApplet.appOutput("Connection Failed!\n");
        }
    }
}

```

Creating a Database File

This section creates a database file in the current account.

```

class CreateFileListener implements ActionListener
{
    // handle to the base applet

    private FileDemo parentApplet;

    // constructor that requires a handle to the base applet
    CreateFileListener(FileDemo handle)
    {
        parentApplet = handle;
    }

    // This method is executed whenever the Create File button is
    // pressed.

    public void actionPerformed(ActionEvent event)
    {
        UniFile demoFile = null;
        UniCommand command = null;
        parentApplet.appOutput("Create File started.\n");
        try
        {
            int status = 0;

            // check if JAVA_DEMO exists, if so, then delete

            try
            {
                demoFile = parentApplet.session.open("JAVA_DEMO");
                demoFile.close();
                parentApplet.appOutput("  Removing existing JAVA_DEMO
& file.\n");
                command = parentApplet.session.command();
                command.setCommand("DELETE.FILE JAVA_DEMO");
                command.exec();
            }
            catch(UniSessionException e)
            {

                // this exception means that the file doesn't exist so

```

```

we
        // can go ahead and create it.
    }
    catch(UniFileException e)
    {
        parentApplet.appOutput("  ERROR:UniFileException:"
            + e.getMessage() + "\n");
    }

    parentApplet.appOutput("  UniCommand =
        + UniSesson.command()\n");

    // create a database command object

    command = parentApplet.session.command();
    parentApplet.appOutput("
UniCommand.setCommand(CREATE.FILE
        + JAVA_DEMO 25 Java Demo File)\n");
    command.setCommand("CREATE.FILE JAVA_DEMO 25 Java Demo
File");
    parentApplet.appOutput("  UniCommand.exec()\n");

    // execute our create file command

    command.exec();

    // print out the results from the execute command

    parentApplet.appOutput("  UniCommand.response() = " );
    parentApplet.appOutput(" command.response() );
    parentApplet.appOutput("  UniCommand.status() = " +
        + command.status() + "\n" );
    parentApplet.appOutput("
UniCommand.getSystemReturnCode() =
        + " + command.getSystemReturnCode() + "\n" );
    }
    catch(UniSessionException e)
    {
        parentApplet.appOutput("  ERROR:UniSessonException:" +
            + e.getMessage() + "\n");
    }
    catch(UniCommandException e)
    {
        parentApplet.appOutput("  ERROR:UniCommandException:" +
            + e.getMessage() + "\n");
    }
    parentApplet.appOutput("Create File completed.\n");
}
}

```

Loading a Database File

This section loads a database file from the current account.

```
class LoadFileListener implements ActionListener
{
    // handle to the base applet

    private FileDemo parentApplet;

    // constructor that requires a handle to the base applet

    LoadFileListener(FileDemo handle)
    {
        parentApplet = handle;
    }

    // This method is executed whenever the Load File button is
    pressed.

    public void actionPerformed(ActionEvent event)
    {
        UniFile demoFile = null;
        String data = null;
        char chr;=20
        parentApplet.appOutput("Load File started.\n");
        try
        {
            parentApplet.appOutput("    UniFile =
                â UniSession.open(JAVA_DEMO)\n");

            // create a new file object

            demoFile = parentApplet.session.open("JAVA_DEMO");

            // if the session object supports encryption, then let's
turn
            // encryption off for this file.

            if(parentApplet.session.isEncryptionEnabled())
            {
                demoFile.setEncryptionType(1);
            }
            chr = 'a';
            parentApplet.appOutput("    UniFile.write(Record ID,
                â Record Data) 200 times!\n");

            // now lets fill the file with meaningless data

            for (int count = 0; count < 200; count++)
            {
                data = data + chr;
                demoFile.write(new Integer(count), data);
            }
        }
    }
}
```

```

        chr++;
    }
    parentApplet.appOutput("    UniFile.close()\n");

    // close the file we opened

    demoFile.close();=20
}
catch(UniSessionException e)
{
    parentApplet.appOutput("    ERROR:UniSessionException:" +
        "\n" + e.getMessage() + ":" + e.getExtendedMessage() +
        "\n");
}
catch(UniFileException e)
{
    parentApplet.appOutput("    ERROR:UniFileException:" +
        "\n" + e.getMessage() + "\n");
}
parentApplet.appOutput("Load File completed.\n");
}
}

```

Creating a Select List

This section creates a select list from the demo file in the current account.

```

class CreateSelectListListener implements ActionListener
{
    // handle to the base applet

    private FileDemo parentApplet;

    // constructor that requires a handle to the base applet

    CreateSelectListListener(FileDemo handle)
    {
        parentApplet = handle;
    }

    // This method is executed whenever the Create Select List
    button is
    // pressed.

    public void actionPerformed(ActionEvent event)
    {
        UniFile demoFile = null;
        String data;
        char chr;
        parentApplet.appOutput("Create Select List started.\n");
        try
        {

```

```

        parentApplet.appOutput("
UniSession.open('JAVA_DEMO')\n");

        // create a new file object

        demoFile = parentApplet.session.open("JAVA_DEMO");

        // if we have an existing select list then lets empty it

        if ( parentApplet.demoSelect != null )
        {
            parentApplet.appOutput("    UniSession.clearList()\n");
            parentApplet.demoSelect.clearList();
        }
        parentApplet.appOutput("    UniSession.selectList(1)\n");

        // create a new select list

        parentApplet.demoSelect =
parentApplet.session.selectList(1);
        parentApplet.appOutput("
UniSelectList.select(UniFile)\n");

        // generate a select list of the file we opened earlier

        parentApplet.demoSelect.select(demoFile);
        parentApplet.appOutput("    UniSelectList.saveList
        å (JAVA_DEMO_LIST)\n");

        // save the list we generated to a file

        parentApplet.demoSelect.saveList("JAVA_DEMO_LIST");
        parentApplet.appOutput("    UniFile.close()\n");

        // close the file object now that we are done with it

        demoFile.close();
    }
    catch(UniSessionException e)
    {
        parentApplet.appOutput("    ERROR:UniSessonException:" +
        å e.getMessage() + "\n");
    }
    catch(UniFileException e)
    {
        parentApplet.appOutput("    ERROR:UniFileException:" +
        å e.getMessage() + "\n");
    }
    catch(UniSelectListException e)
    {
        parentApplet.appOutput("    ERROR:UniSelectListException:"

```

```

+
        a e.getMessage() + "\n");
    }
    parentApplet.appOutput("Create Select List completed.\n");
}
}

```

Reading and Displaying the Select List

This section reads and displays the select list of the demo file in the current account.

```

class ShowSelectListListener implements ActionListener
{
    // handle to the base applet

    private FileDemo parentApplet;

    // constructor that requires a handle to the base applet

    ShowSelectListListener(FileDemo handle)
    {
        parentApplet = handle;
    }

    // This method is executed whenever the Show Select List button
is
    // pressed.

    public void actionPerformed(ActionEvent event)
    {
        UniFile demoFile = null;
        String data;
        char chr;=20
        parentApplet.appOutput("Show Select List started.\n");
        try
        {

            // check to see if we have created a select list during
the
            // life of this applet

            if (parentApplet.demoSelect != null)
            {
                parentApplet.appOutput( "    UniSelectList.getList
                a (JAVA_DEMO_LIST)\n" );
                // load the select list we previously generated

                parentApplet.demoSelect.getList("JAVA_DEMO_LIST");
                parentApplet.appOutput( "
                UniSelectList.readList()\n" );

                // read the list into a string and then output it

```

```

UniString result = parentApplet.demoSelect.readList();
parentApplet.appOutput( " " + result.toString() +
"\n" );

UniDynArray dynResult = new UniDynArray(result);
parentApplet.appOutput( " The English version!\n" );

// output the select list in a formatted form

for(int i = 0;i < dynResult.count();i++)
{
    parentApplet.appOutput( " " +
    dynResult.extract(i).toString() + "," );
    if( ((i+1) % 10) == 0)
    {
        parentApplet.appOutput( "\n" );
    }
}
else
{
    parentApplet.appOutput(
" If you want to look at the select list, then
you    dynResult must first create one!\n");
}
catch(UniSelectListException e)
{
    parentApplet.appOutput(" ERROR:UniSelectListException:"
+
    e.getMessage() + "\n");
}
parentApplet.appOutput("Show Select List completed.\n");
}
}

```

Listing the File

This section reads and displays the contents of the demo file.

```

class ListFileListener implements ActionListener
{
    // handle to the base applet

    private FileDemo parentApplet;

    // constructor that requires a handle to the base applet

    ListFileListener(FileDemo handle)
    {
        parentApplet = handle;
    }
}

```

```

    }

    // This method is executed whenever the List File button is
    // pressed.

    public void actionPerformed(ActionEvent event)
    {
        UniFile demoFile = null;
        String data;
        char chr;=20
        parentApplet.appOutput("List File started.\n");
        try
        {
            if (parentApplet.demoSelect == null)
            {
                parentApplet.appOutput("   If you want to run this
command      å you must first press the Create Select List
              å button.\n");
            }
            else
            {
                // we previously generated a select list so now lets
use it

                parentApplet.appOutput( "
UniSelectList.getList(JAVA_DEMO_LIST)\n" );
                // load the select list we previously generated

                parentApplet.demoSelect.getList("JAVA_DEMO_LIST");
                parentApplet.appOutput("
UniSession.open('JAVA_DEMO')\n");

                // create a new file object

                demoFile = parentApplet.session.open("JAVA_DEMO");

                parentApplet.appOutput("   UniSelectList.next()\n");
                parentApplet.appOutput("   UniFile.read(Record
ID)\n");

                // list the entire contents of the file
                // read first recordID

                UniString recordID = parentApplet.demoSelect.next();
                while ( !parentApplet.demoSelect.isLastRecordRead() )
                {
                    UniString record = demoFile.read(recordID);
                    parentApplet.appOutput(demoFile.getRecordID() +
"\t" +
                    å demoFile.getRecord() + "\n");

                    // read next record ID

```

```

        recordID = parentApplet.demoSelect.next();
    }

    // close the file we opened

    demoFile.close();
    parentApplet.appOutput("    UniFile.close()\n");
}
}
catch(UniSessionException e)
{
    parentApplet.appOutput("    ERROR:UniSessonException:" +
        "\n" + e.getMessage() + "\n");
}
catch(UniFileException e)
{
    parentApplet.appOutput("    ERROR:UniFileException:" +
        "\n" + e.getExtendedMessage() + "\n");
}
catch(UniSelectListException e)
{
    parentApplet.appOutput("    ERROR:UniSelectListException:"
+
        "\n" + e.getMessage() + "\n");
}
parentApplet.appOutput("List File completed.\n");
}
}

```

Executing a Command

This section executes a database command in the current account and displays the output.

```

class ExecuteCommandListener implements ActionListener
{
    // handle to the base applet

    private FileDemo parentApplet;

    // constructor that requires a handle to the base applet

    ExecuteCommandListener(FileDemo handle)
    {
        parentApplet = handle;
    }

    // This method is executed whenever the Execute Command button
    is
    // pressed.

    public void actionPerformed(ActionEvent event)

```

```

{
    UniCommand command = null;
    parentApplet.appOutput("Execute Command started.\n");
    try
    {
        // create a command object

        command = parentApplet.session.command();
        parentApplet.appOutput( "    UniCommand.setCommand(LIST
VOC
        â SAMPLE 10)\n");

        // set the TCL text of the command you wish to execute

        command.setCommand("LIST VOC SAMPLE 10");
        parentApplet.appOutput( "    UniCommand.exec()\n");

        // execute your command

        command.exec();
        parentApplet.appOutput( "    UniCommand.response() = " );

        // display the results of the command

        parentApplet.appOutput( command.response() );
        parentApplet.appOutput( "    UniCommand.status() = " +
            â command.status() + "\n" );
        parentApplet.appOutput( "
UniCommand.getSystemReturnCode() =
            â " + command.getSystemReturnCode() + "\n" );
    }
    catch(UniSessionException e)
    {
        parentApplet.appOutput( "    ERROR:UniSessionException:" +
            â e.getMessage() + "\n");
    }
    catch(UniCommandException e)
    {
        parentApplet.appOutput( "    ERROR:UniCommandException:" +
            â e.getMessage() + "\n");
    }
    parentApplet.appOutput("Execute Command completed.\n");
}
}

```

Executing a Subroutine

This section executes a BASIC subroutine in the current account and displays the output.

```
class ExecuteSubroutineListener implements ActionListener
```

```

{
    // handle to the base applet

    private FileDemo parentApplet;

    // constructor that requires a handle to the base applet

```

```

ExecuteSubroutineListener(FileDemo handle)
{
    parentApplet = handle;
}

// This method is executed whenever the Execute Subroutine
button is
// pressed.

public void actionPerformed(ActionEvent event)
{
    UniSubroutine subroutine = null;
    parentApplet.appOutput("Execute Subroutine started.\n");
    try
    {
        parentApplet.appOutput("    UniSubroutine =
            a UniSesson.subroutine(!TIMDAT, 1)\n");

        // create a subroutine object

        subroutine =
parentApplet.session.subroutine("!TIMDAT",1);
        parentApplet.appOutput("    UniSubroutine.call()\n");

        // execute the specified subroutine

        subroutine.call();
        String outputStr = subroutine.getArg( 0 );
        UniDynArray outputDyn = new UniDynArray(outputStr);
        parentApplet.appOutput("    UniSubroutine.getArg(0) = " +
            a outputStr + "\n");
        // display the results of the subroutine

        parentApplet.appOutput("    The English version of the
            a subroutine return argument!\n" );
        parentApplet.appOutput("    Month=" + outputDyn.extract(0)
+
            a "\n");
        parentApplet.appOutput("    Day of Month=" +
            a outputDyn.extract(1) + "\n");
        parentApplet.appOutput("    Year=" + outputDyn.extract(2)
+
            a "\n");
        parentApplet.appOutput("    Minutes since midnight=" +
            a outputDyn.extract(3) + "\n");
        parentApplet.appOutput("    Seconds into the minute=" +
            a outputDyn.extract(4) + "\n");
        parentApplet.appOutput("    Ticks of last second since
            a midnight=" + outputDyn.extract(5) + "\n");
        parentApplet.appOutput("    CPU second used since entering
the
            a database=" + outputDyn.extract(6) + "\n");
        parentApplet.appOutput("    Ticks of last second used

```

```

since
    â login=" + outputDyn.extract(7) + "\n");
parentApplet.appOutput("    Disk I/O seconds used since
    â entering the database=" + outputDyn.extract(8) +
    â "\n");
used
parentApplet.appOutput("    Ticks of last disk I/O second
    â since login=" + outputDyn.extract(9) + "\n");
parentApplet.appOutput("    Number of ticks per second=" +
    â outputDyn.extract(10) + "\n");
parentApplet.appOutput("    User number=" +
    â outputDyn.extract(11) + "\n");
parentApplet.appOutput("    Login ID=" +
    â outputDyn.extract(12) + "\n");
}
catch(UniSessionException e)
{
    parentApplet.appOutput("    ERROR:UniSessonException:" +
    â e.getMessage() + "\n");
}
catch(UniSubroutineException e)
{
    parentApplet.appOutput("    ERROR:UniSubroutineException:"
+
    â e.getMessage() + "\n");
}
parentApplet.appOutput("Execute Subroutine completed.\n");
}
}

```

Disconnecting from the Database

This section disconnects from the database.

```

class DisconnectListener implements ActionListener
{
    // handle to the base applet

    private FileDemo parentApplet;

    // constructor that requires a handle to the base applet

    DisconnectListener(FileDemo handle)
    {
        parentApplet = handle;
    }

    // This method is executed whenever the Disconnect button is
    // pressed.

    public void actionPerformed(ActionEvent event)
    {

```

```

        parentApplet.appOutput("Disconnecting from Database
Server.\n");

        // check to make sure we originally established a session

        if(parentApplet.session != null)
        {
            try
            {
                parentApplet.appOutput("
UniSession.disconnect()\n");

                // disconnect from the server

                parentApplet.session.disconnect();
                parentApplet.demoSelect = null;
            }
            catch (UniSessionException e)
            {
                parentApplet.appOutput("  ERROR:UniSessionException:"
+
                "  " e.getMessage() + "\n");
            }
        }
        parentApplet.appOutput("Connection Successfully
terminated.\n");
    }
}

```

Exiting FileDemo

This section exits *FileDemo*. It is used only when *FileDemo* is launched as an application. If it is launched as an applet, the **Exit Demo** button is not displayed.

```

class ExitDemoListener implements ActionListener
{
    // handle to the base applet

    private FileDemo parentApplet;

    // constructor that requires a handle to the base applet

    ExitDemoListener(FileDemo handle)
    {
        parentApplet = handle;
    }

    // This method is executed whenever the Exit Demo button is
    pressed.

    public void actionPerformed(ActionEvent event)
    {

```

```

parentApplet.appOutput("Exiting Demo Applet.\n");

// check to make sure we originally established a session

if( parentApplet.session != null)
{
    try
    {
        // disconnect from the server

        parentApplet.session.disconnect();
        parentApplet.demoSelect = null;
    }
    catch (UniSessionException e)
    {
        parentApplet.appOutput("  ERROR:UniSessionException:"
+
        "  " e.getMessage() + "\n");
    }
}
System.exit(0);
}

```

Index

A

administering the proxy server 3-8

B

blockingStrategy method 2-18

C

call method 2-26, 4-18, 4-150
cancel method 4-18, 4-145
case-sensitivity 4-5
CATALOG command 2-25
catalog space 2-25
cataloged subroutine 2-8
clearFile method 4-18
CLEARFILE statement 4-18
clearList method 2-20, 4-18, 4-138
client, calling server subroutines
 from 2-26
client/server programs 2-25
CLOSE statement 4-18
closeFile method 4-18
closeSeqFile method 4-97
code portability 4-46
command method 2-11, 2-24
commandStatus method 2-24
commands, proxy server 3-9
commit method 4-18, 4-155
connect method 2-10, 4-18
conversion codes 2-16
 in file dictionary 2-5
converting data 2-5
count method 4-18, 4-120, 4-121
CREATE.FILE command 2-26

D

data conversions
 iconv method 2-16
 oconv method 2-16
 using file dictionary 2-5
data retrieval 2-6
D-descriptors 2-5
defaultBlockingStrategy method 2-19
defaultLockStrategy method 2-19
defaultReleaseStrategy method 2-19
del method 4-18, 4-121
DELETE statement 4-18
deleteRecord method 4-18
DELETEU statement 4-18
delimiters A-17
demo application B-1
 code structure B-4
 installing B-2
dictionaries 2-5
 types of record in 2-5
 using 2-20
disconnect method 4-18
documentation conventions 1-x
dynArray method 2-11

E

editing proxy server configuration
 file 3-3
Editor 2-6
error method 2-24
errors
 codes A-1
 handling in Visual Basic 2-17
 replace tokens A-2

exec method 2-24, 4-18, 4-145
EXECUTE statement 2-24

F

field method 4-18
file dictionaries 2-5
 types of record in 2-5
file pointers, moving 2-22
FILELOCK statement 4-19
files
 for storing text or binary data 2-22
 opening to a variable 2-12
 type 1 2-22
 type 19 2-22
fileSeek method 2-22, 4-18, 4-98
FILEUNLOCK statement 4-21
formatting data 2-5
formList method 2-20, 4-18, 4-138

G

getAkInfo method 4-19
getArg method 2-26, 4-19, 4-151
getAssoc method
 of UniDictionary object 4-74, 4-85
getAtVariable method 4-19
getBlockingStrategy method
 of UniFile object 4-54, 4-63, 4-75, 4-86
 values A-9
getBlockSize method of UniCommand object 4-147
getClientNames method of UniNLSLocale object 4-158
getClientNames method of UniNLSMap object 4-160
getConv method
 of UniDictionary object 4-75, 4-86
getFileName method of UniFile object 4-55, 4-64, 4-76, 4-87
getFileType method of UniFile object 4-55, 4-76
getFormat method
 of UniDictionary object 4-77, 4-87
getList method 4-19, 4-139
getLoc method
 of UniDictionary object 4-77, 4-88

getLockStrategy method
 of UniFile object 4-56, 4-64, 4-78, 4-88
getName method
 of UniDictionary object 4-78, 4-89
getPassword method of UniSession object 4-46
getReadSize method of UniSequentialFile object 4-99, 4-102
getRecord method of UniFile object 4-56, 4-64, 4-78, 4-89
getRecordID method of UniFile object 4-56, 4-64, 4-79, 4-89
getReleaseStrategy method
 of UniFile object 4-56, 4-65, 4-79, 4-89
getRoutineName method of UniSubroutine object 4-153
getServerName method of UniNLSMap object 4-161
getServerNames method of UniNLSLocale object 4-159
getSM method
 of UniDictionary object 4-80, 4-91
getSQLType method
 of UniDictionary object 4-80, 4-91
getTimeout method of UniSession object 4-48
getTransport method of UniSession object 4-48
getType method
 example using 2-21
 of UniDictionary object 4-81, 4-91
getUserName method of UniSession object 4-48
global constants A-1
granting access to the proxy server 3-5

H

help for the proxy server 3-10

I

iconv method 4-19
IDEAL flavor accounts 2-4, 5-6
I-descriptors 2-6

 evaluated by readNamedField method 2-16
INDICES statement 4-19
ins method 4-19, 4-123
installation
 demo application B-2
isActive method 4-19, 4-156
isOpen method 4-19, 4-99
 and UniFile object 4-58, 4-81
iType method 4-19
iType method and UniFile object 4-58, 4-82

L

length method 4-19, 4-123
lockFile method 4-19, 4-59, 4-82
lockRecord method 4-19, 4-59, 4-82
locks 2-6
 and UniObjects 2-18
 overview 2-18
 releasing at end of session 2-19
 setting and releasing 2-18
 setting default locking action 2-19
lockStrategy method 2-19

M

methods
 definition 2-8
 equivalent BASIC statements 2-8
 equivalent UniObjects methods 2-8
 quick reference 4-7
moving file pointers 2-22
multivalued fields 2-5

N

network, reducing traffic 2-25
next method 2-20, 4-19, 4-139
nextBlock method 2-24, 4-19
NLS (National Language Support)
 conventions 4-158
 UniNLSLocale object 4-158
 UniNLSMap object 4-160
 with UniObjects for Java 1-3

O

objects
 definition 2-7
 quick reference 4-7
 UniCommand 2-8, 4-144
 UniDictionary 2-7, 4-71
 UniDynArray 2-7, 4-119
 UniFile 2-7, 4-51
 UniNLSLocale 4-158
 UniNLSMap 4-160
 UniSelectList 2-7, 4-137
 UniSequentialFile 2-7, 4-97
 UniSession 2-7, 4-25
 UniSubroutine 2-8, 4-150
 UniTransaction 4-155
 used in UniVerse 2-7

oconv method 4-19

open method 2-11, 2-12
 with text and binary files 2-22

openDict method 2-11, 2-21, 4-19

openSeq method 2-11, 2-22, 4-40

overview
 of locks 2-18
 of UniObjects for Java 1-3
 of UniVerse 2-4

P

parameters, proxy server 3-5

proxy server 3-2
 administering 3-8
 commands 3-9
 editing the configuration file 3-3
 getting help for 3-10
 granting access 3-5
 parameters 3-5
 setting up 3-3
 starting 3-7

Q

quick reference, objects and
 methods 4-7

R

read method 2-13, 2-22, 4-19, 4-60, 4-83

READ statement 4-19

readBlk method 2-22, 4-19, 4-100

readField method 4-19, 4-61, 4-84

READL statement 4-19

readLine method 2-22, 4-19, 4-100

readList method 2-11, 2-20, 4-20, 4-139

readNamedField method 2-16, 4-20, 4-62, 4-85

READU statement 4-19

READV statement 4-19

READVL statement 4-19

READVU statement 4-19

RECORDLOCKL statement 4-19

RECORDLOCKU statement 4-19

records
 modifying a named field 2-16
 writing 2-13

REFORMAT command 2-26

relative position parameter values A-14

RELEASE statement 4-21

releaseStrategy method 2-19

releaseTaskLock method 4-20

replace method 4-20, 4-124

replace tokens A-2

reply method 2-24, 4-20, 4-147

resetArgs method 4-20, 4-152

response method 2-24
 of UniCommand object 4-147

Retrieve processor 2-6

ReVis processor 2-6

rollback method 4-20, 4-156

S

saveList method 4-20, 4-140

SELECT command 2-20, 2-26

select lists 2-20
 accessing 2-20
 creating 2-20

select method 2-20, 4-20

selectAlternateKey method 2-20, 4-20, 4-141

selectList method 2-11, 2-20, 4-20
 description 4-41

selectMatchingAk method 2-20, 4-20, 4-141

server
 connecting with 2-10
 running subroutines on 2-25

setArg method 2-26, 4-20, 4-152

setAssoc method 2-21
 of UniDictionary object 4-74, 4-85

setAtVariable method 4-20

setBlockingStrategy method
 of UniFile object 4-54, 4-63, 4-75, 4-86
 values A-9

setBlockSize method of UniCommand
 object 4-147

setClientNames method of
 UniNLSLocale object 4-158

setClientNames method of
 UniNLSMap object 4-160

setCommand method 2-24
 of UniCommand object 4-148

setConv method 2-21
 of UniDictionary object 4-75, 4-86

setFileName method of UniFile
 object 4-55, 4-64, 4-76, 4-87

setFileType method of UniFile
 object 4-55, 4-76

setFormat method 2-21
 of UniDictionary object 4-77, 4-87

setLoc method 2-21
 of UniDictionary object 4-77, 4-88

setLockStrategy method
 of UniFile object 4-56, 4-64, 4-78, 4-88

setName method 2-21, 4-20
 of UniDictionary object 4-78, 4-89

setPassword method of UniSession
 object 4-46

setReadSize method of
 UniSequentialFile object 4-99, 4-102

setRecord method of UniFile object 4-56, 4-64, 4-78, 4-89

setRecordID method of UniFile
 object 4-56, 4-64, 4-79, 4-89

setRecordID method, assigning a new
 value to 2-19

setReleaseStrategy method
of UniFile object 4-56, 4-65, 4-79, 4-89

setRoutineName method of
UniSubroutine object 4-153

setServerName method of UniNLSMap
object 4-161

setServerNames method of
UniNLSLocale object 4-159

setSM method 2-21
of UniDictionary object 4-80, 4-91

setSQLType method 2-21
of UniDictionary object 4-80, 4-91

setTaskLock method 4-20

setTimeout method of UniSession
object 4-48

setting up the proxy server 3-3

setTransport method of UniSession
object 4-48

setType method 2-21
of UniDictionary object 4-81, 4-91

setUserName method of UniSession
object 4-48

SSELECT command 2-20

start method 4-20, 4-156

starting the proxy server 3-7

status method
of UniFile object 4-66, 4-92
of UniSequentialFile object 4-102

subroutine method 2-11, 2-25, 4-20

subroutines
running on the server 2-25
supplying arguments to 2-26

subValue method 4-21

system delimiters A-17

T

terminology 2-7

toString method of UniDynArray
object 4-125

TRANS function 2-6

transaction method of UniSession
object 4-49

U

UniCommand object 2-8

description 4-144

using 2-24

UniDataSet object 4-129

UniDictionary object 2-7, 2-21
description 4-71

UniDynArray object 2-7
description 4-119

UniFile object 2-7, 2-12, 4-51
description 4-51
program example 4-70

UniNLSLocale object, description 4-158

UniNLSMap object, description 4-160

UniObjects for Java
features 1-5
overview 1-3

UniObjects, equivalents to methods 2-8

UniSelectList object 2-7, 2-20
description 4-137

UniSequentialFile object 2-7, 4-97
accessing 2-22
description 4-97
example 4-104

UniSession object 2-7, 4-25
connect method 2-10
description 4-25
program example 4-49

UniStringTokenizer object 4-117

UniSubroutine object 2-8, 2-25
description 4-150

UniTransaction object, description 4-155

UniVerse
account flavors 2-4, 5-6
data retrieval 2-6
data structure 2-4
ending a session 2-10
environment 2-4
executing commands 2-24
file dictionaries 2-5
multivalues 2-5
opening a session 2-10
using commands 2-26
using files in 2-12
VOC file 2-4

UniVerse BASIC, equivalents to
statements 2-8

UniVerse NLS, *see* NLS

unlockFile method 4-21, 4-66, 4-92

unlockRecord method 2-19, 4-21, 4-66, 4-92

V

value method 4-21, 4-125

variables
Object type 2-12
@ A-8
@TTY 2-11

VOC file 2-4

W

write method 2-13, 4-21, 4-67, 4-93

WRITE statement 4-21

writeBlk method 2-22, 4-21, 4-102

writeEOF method 2-22, 4-21, 4-103

writeField method 4-21, 4-68, 4-94

writeLine method 2-22, 4-21, 4-103

writeNamedField method 2-16, 4-21, 4-69, 4-95

WRITEU statement 4-21

WRITEV statement 4-21

WRITEVU statement 4-21

writing records 2-13

Symbols

@TTY variable 2-11

@variables A-8