Adabas Client for Java

Adabas Client for Java API

Version 2.2

October 2021

ADABAS & NATURAL

This document applies to Adabas Client for Java Version 2.2 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: ACJ-API-22-20210929

Table of Contents

Preface

This documentation contains information about the main concepts involved in the Adabas Client for Java. It also describes the main classes used, and how to add the example programs into Eclipse.

The following topics are covered:

Adabas Client for Java API	Explains the main concepts behind Adabas Client for Java API.
Use Cases	Common use cases.
Adabas Client for Java Main Classes	Descriptions of the main classes used by Adabas Client for Java.
Adabas Client for Java and Eclipse	How to use Adabas Client for Java with Eclipse.

Note: The Installation directory of Adabas Client for Java contains a set of JavaDocs for Adabas Client for Java, as well as example Java programs.

1 About this Documentation

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at **https://documentation.softwareag.com**.

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to empower@softwareag.com with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at **https://empower.softwareag.com/**.

You can find product information on the Software AG Empower Product Support website at **https://empower.softwareag.com**.

To submit feature/enhancement requests, get information about product availability, and download products, go to **Products**.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the **Knowledge Center**.

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at **https://empower.softwareag.com/public_directory.aspx** and give us a call.

Software AG Tech Community

You can find documentation and other technical information on the Software AG Tech Community website at **https://techcommunity.softwareag.com**. You can:

- Access product documentation, if you have Tech Community credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

2 Adabas Client for Java API

The Adabas Client for Java API is used to introduce Adabas functionality into the Java programming language.

Adabas Mapping

Adabas Client for Java maps long names to the Adabas short names in the FDTs. The main advantages of this include:

- The map configuration is stored in an Adabas file which is part of the database. The mapping can be accessed from everywhere, even remotely.
- Various input formats can be used as the source of the map definitions. It is possible to use SYSTRANS files based on Natural DDMs to define the map; FDT comments and XML Schema files (XSD) can also be used.
- Data can be accessed without having to know the database ID and file number, only the map name is required.

The API provides a class called `AdabasMapper`, which performs all read and write operations out of and into the map configuration. The `AdabasMapper` also provides search functionality to find maps.

The `AdabasMapper` class produces so-called Adabas targets and Adabas sessions which are described below.

Adabas Target

The Adabas target defines the connection to the target database. This might be just a database ID, or, in case of mainframe databases where the Entire Net-Work port number is known, the target is a combination of the database ID and Entire Net-Work connection URL. The Software AG Directory Server is the reference for remote databases.

The Adabas target also contains a number of connection parameters, such as the host's endianness or the character set used. The Adabas target handles the current connection state.

Adabas Session

An Adabas session contains all Adabas user-specific information. This includes the user ID, process identification (EID) and some security-related information, such as the RACF or LUW Adabas Security credentials.

Adabas Client Java Session

An Adabas Client Java Session is essentially defined by the two parameters Adabas Target (`AdabasTarget.class`) and Adabas Session (`AdabasSession.class`). These two objects are created either directly, using the specific `AdabasTarget` or `AdabasSession classes`, or they can be created using the `AdabasConnection` helper class.

The `AdabasConnection` class is created using an SQL-like connection string. With an `AdabasConnection` instance you can create request instances which handle specific read, delete, update or write operations on the database. Each request contains information that is specific to the operation it provides. Multiple requests can be used per session. Multiple requests need to be used to read a different set of fields out of an Adabas database. If a number of Adabas files are accessed in an Adabas session, each access need to be distributed in several request classes. Internally, each request uses a format buffer and other Adabas-specific objects, which are generated on a per request basis.

Transaction

A transaction consists of a chain of request operations. Any number of transactions can be part of an Adabas session.

Request Chain Example

The following is an example of how a request chain might look:

```
AdabasTarget target = new AdabasTarget(getDynamicDbid());
 try {
   /* Setting the target to not implicit close the request provides the possiblity ↩
for multiple
      request in one Adabas session  */
   target.setImplicitClose(false);
   /* Read target database on file 11 field "AA", "AB" */
   try (ReadRequest request = new ReadRequest(target, 11)) {
       request.addFieldsQuery(new String[] { "AA", "AB" });
       QueryResultList list = (QueryResultList) request.readIsnSequence();
   }
   /* Store records in file 17 and file 16 with different fields dependent on each ↩
file */
   try (StoreRequest storeRequest1 = new StoreRequest(target, 17)) {
      try (StoreRequest storeRequest2 = new StoreRequest(target, 16)) {
         /* Create a new record with corresponding fields and fill value */
        RecordEntry entry = storeRequest1.createRecordEntry(new String[] { "AA", ↩
"AE" });
         entry.addValue("AA", 123);
         /* Store entry - Dependent on the cache the store is not send to the ↩
database */
         storeRequest1.storeEntry(entry);
         try (ReadRequest request = new ReadRequest(target, 11)) {
             request.addFieldsQuery(new String[] { "AA", "AW" });
             QueryResultList list = (QueryResultList) request.readIsnSequence();
             list.output(System.out);
         } /* Close here is omitted */
        /* Create a new record for file 16 with corresponding fields and fill value ↩
*/
         RecordEntry entry = storeRequest2.createRecordEntry(new String[] { "AA", ↩
"AE" });
         entry.addValue("AA", String.format("%06d", i));
         storeRequest2.storeEntry(entry);
         /* Flush records to database file 16 */
         storeRequest2.endTransaction();
         /* Flush records to database file 17*/
         storeRequest1.endTransaction();
       }
    } /* Pay attention, a implict ET is done here */
    /* Read with the same connection target the file 16 after the Transaction */
    try (ReadRequest request = new ReadRequest(target, 16)) {
      request.addFieldsQuery(new String[] { "AA", "A1", "AQ", "AW" });
      QueryResultList list = (QueryResultList) request.readIsnSequence();
      list.output(System.out);
     }
   } finally {
     target.close();
   }
```

An Adabas session can be reused after the current Adabas target connection is being closed; the target will open automatically if a new request is initiated on the target. If it is used for map-based

access, the AdabasConnection or the AdabasMapper instance will be used instead of the target and the file number.

Adabas Session Authentication Types

Adabas Client for Java supports the following Adabas session types:

Type	Description
NONE	The default Adabas ID is generated containing the user ID, host and an additional process identification (so-called ESID) describing the uniqueness.
ADA_SECURITY	The Adabas security password is sent to the database. Refer to *Adabas Security* in the Adabas documentation for further information.
ADA_PW_SECURITY	This session type supports the new Adabas Security on Linux, UNIX and Windows platforms. In addition to the Adabas ID, information like the user ID and pasword are required.
ADASAF	This session type is used to send credentials to a remote mainframe database, which is secured using ADASAF and the external security system RACF.

3 Use Cases

The following chapter presents some common use cases for Adabas Client for Java.

Connecting to an Adabas Database Engine

The Adabas target can be a local database, or it can be a remote database connection path.

Local Database

A local Adabas database target is accessed using the classic ADALNK libraries to send Adabas calls. In this case, only a database number must be specified.

```
/* simple Adabas Database target definition (dbid) */
AdabasTarget target = new AdabasTarget(dbid);
target.open();
```

Remote Database

A remote Adabas database is referenced using an Entire Net-Work (WCP) URL connection string.

```
AdabasTarget(int dbid, java.lang.String url)
Adabas Target definition with an Entire Net-Work remote URL reference.
```

Sample URL reference: `tcpip://<hostname>.<domainname>:<port>`

On mainframe platforms, Adabas Client for Java API uses the EBCDIC (037) character set to connect to and communicate with an Adabas server environment. Software AG recommends that you use UES-enabled mainframe databases in order to avoid this restriction.

Note: It is also possible to use an Adabas Directory Server or a flat file named *xtsurl.cfg*, that contains remote database URL references.

Read Adabas Data

In addition to the database ID, additional parameters, such as the file number and short name fields, need to be specified. The main class used to read data from Adabas is the `ReadRequest` class.

A simple example application (which accesses the fields AA and AB of file number 11 in database 24) would require the following settings:

```
/* Short name fields to generate the Adabas Format Buffer */

    String fields = new String{"AA","AB"};

    /* Create read request with parameter database id and file number */

   ReadRequest request = new ReadRequest(24,11).queryFields(fields);

   /* Send request and receive results */

   QueryResultList list = (QueryResultList) request.readIsnSequence();
```

The `readIsnSequence()` method sends the query to the database and the results will be collected in a list array. The list contains entries for each database record.

The following example will return the field value instance of field "AC" in the first record:

```
IRecordEntry record = list.get(0);

   IAdaFieldValue fieldValueFirstName = record.getDeepFieldValuebyShortName("AC");

   String firstName = fieldValueFirstName.getValue(); ↩
```

Note: By default, database security is disabled, and therefore no session information is required for further initialization. As soon as the database is protected, either by authentication and/or by authorization rules, session information is required - refer to the `ReadRequest` constructors and `AdabasSession` class for further information.

Read Adabas Data using Map Definition

The mapping extension allows you to use long names instead of short names (2 byte) to reference Adabas fields. The corresponding metadata is stored in an Adabas database file (map configuration file). It is recommended that you use one configuration file per database.

Refer to the description of the Data Designer for further details about mapping.

The location of the map definition must be registered. This can be done using the `AdabasMapper` or `ReadRequest` classes:

- `AdabasMapper`

```
AdabasMapper.addMapStorage(<URL location>,<configuration file number> );
```

- `ReadRequest`

```
ReadRequest request = new ReadRequest("EmployeeMap",<url to database>,<map ↩
configuration file>).queryFields(fields);
```

The main class used to read data from Adabas is the `ReadRequest` class. The `ReadRequest` can be initialized with just the classic parameters. A simple example application requires the following statements:

```
/* Long name fields now possible */
    String fields = new String{"FirstName","LastName"};

    /* Create request containing needed parameters */
   ReadRequest request = new ReadRequest("EmployeeMap").queryFields(fields);

   /* Send request and receive result */
   QueryResultList list = (QueryResultList) request.readIsnSequence();
```

This example uses the map named `EmployeeMap`. The database ID and file number are part of the mapping information. In this example, the map defines database ID 24 and the file number 11 as the data location. File 11 is the standard employee demo file delivered with Adabas on all Linux, Unix and Windows platforms. The Adabas fields are listed in the “fields” variable. The field `FirstName` is mapped to AC and the field `LastName` is mapped to AE.

The following statements return the field value instance of field `LastName` in the first record:

```
IRecordEntry record = list.get(0);
  IAdaFieldValue fieldValueFirstName = record.getDeepFieldValueby("LastName");
  String firstName = fieldValueFirstName.getValue(); ↩
```

Note: Please look at the examples folder (mapping) delivered with the package. Compiling and executing *GenerateMappingsExample* will add two example mapping definitions named `EmployeeMap` and `VehicleMap`, which are used in the other examples. See also the tutorial *Adding the Adabas Client for Java Example Programs in Eclipse*.

Simple Search Example

The `ReadRequest` class includes extensions to support the search capabilties of Adabas.

Simple Search

The Adabas Client for Java API includes the method `setSearch`. The following table shows some example search queries.

Search	Result
AE='SMITH'	Search for records with field AE and value SMITH
AE>'ADAM'	Search for records with field AE and values greater than ADAM
AA>12345	Search for records where field AA is greater than 12345

You only need to add an additional statement to your read request :

```
String fields = new String{"AA","AB"};
  /* Create request containing needed parameters */
  ReadRequest request = new ReadRequest(24,11).queryFields(fields);
  request.setSearch("AE='SMITH'");
  /* Send request and receive result */
  QueryResultList list = (QueryResultList) request.readIsnSequence();
```

Adabas-specific Search

A further method is to generate a search tree. Instead of using the `setSearch(...)` method, the application can generate search trees to support further Adabas search capabilities. For example, it is possible to define values for lower and upper limits:

```
RecordDefinition definition = request.addFieldsQuery(fields);
  /* Get field type for field AE */
  AdaFieldType lastName = definition.getDeepFieldbyShortName("AE");
  /* Define search tree, one node for lower limit and one node for upper limit */
  SearchTree lowerTree = new SearchTree("GE", (IAdaFieldValue) ↩
lastName.getFieldValue());
  lowerTree.setValue("SCHNEIDER".getBytes());
  SearchTree upperTree = new SearchTree("LT", (IAdaFieldValue) ↩
lastName.getFieldValue());
  upperTree.setValue("SD".getBytes());
  lowerTree.bound(upperTree, SearchTree.AND);
  request.setSearchTree(lowerTree);
```

This example will extract only those records that meet the specified criterion of AE >= "SCHNEIDER" and AE < "SD".

Advanced Search Example using Super Descriptor

This example is also intended to illustrate the relationship between the Data Designer and the variable declarations in the java example programs provided - it shows screen shots from the Data Designer together with variable declarations and statements from the example *Superdescriptor-SearchExample.java*.

Group field AB.

Field	Level	Length	Type	Flags
AB	1		Group	
AC	2	20	Alpha	Null Value Suppression
AE	2	20	Alpha	Descriptor
AD	2	20	Alpha	Null Value Suppression

Super descriptor field S2.

Field	Level	Length	Type	Flags
S2	1	26	Super Descriptor	Descriptor
AO	2	1-6	Alpha	
AE	2	1-20	Alpha	

Variable declaration from example *SuperdescriptorSearchExample.java*.

```
private static final int ADA_FILENR = 11;

private static final String ADA_DATA_GROUP = "AB";

private static final String ADA_SEARCH_FIELD = "S2";

private static final String[] ADA_DATA_FIELDS =
    { ADA_SEARCH_FIELD, ADA_DATA_GROUP };

private static final String ADA_SEARCH_VALUE = "'SALE02' 'SMITH'";

private static final String ADA_SEARCH_VALUE_TO = "'SALE02' 'ZZ'";

private static final int START_ISN = 1;

private static final int MAX_RECORDS = 1000;
```

Statements from example *SuperdescriptorSearchExample.java*.

```
try {
    /* simple Adabas Database target definition (dbid) */
    AdabasTarget target = new AdabasTarget(dbid);
    target.open();

    /* Create read request using the database target file number */
    ReadRequest request = new ReadRequest(target, ADA_FILENR);

    /* Set list of Adabas short name fields for read request */
    request.queryFields(ADA_DATA_FIELDS);

    /*
     * Define range for the result set. Set start ISN offset and a
     * maximum value of records to return.
     * Set search criteria - i.e.:  S2 >= "'SALE02' 'SMITH'" &&
     *                              S2 <= "'SALE02' 'ZZ'"
     */
    request.setStart(START_ISN);
    request.setLimit(MAX_RECORDS);

    SearchTree superDescriptorSearchS2 = request.createSearchNode(
        ADA_SEARCH_FIELD, SearchTree.C.GE, ADA_SEARCH_VALUE);
    SearchTree superDescriptorSearchS2To = request.createSearchNode(
        ADA_SEARCH_FIELD, SearchTree.C.LE, ADA_SEARCH_VALUE_TO);
    superDescriptorSearchS2.bound(superDescriptorSearchS2To,
        SearchTree.Logic.AND);
    request.setSearchTree(superDescriptorSearchS2);

    /* Send request and receive result in ISN order */
    QueryResultList list = (QueryResultList) request.read();

    /* Use an internal output method to output data */
    list.output(System.out);
    if (list.size() > 0) {
        /*
         * first value in list :
         * IAdaFieldValue fieldValue =
         * list.get(0).valueOf(ADA_SEARCH_FIELD);
         */
        System.out.println("search criteria -> " + ADA_SEARCH_FIELD
            + " >= '" + ADA_SEARCH_VALUE + "' && " + ADA_SEARCH_FIELD
            + " <= '" + ADA_SEARCH_VALUE_TO + "'");
    }
    request.close();
} catch (QueryException e) {
    System.out.println(e.getMessage());
}
```

4 Main Classes

Adabas Client for Java uses a small set of classes to read or write data into or out of an Adabas database. The following table provides an overview of these main classes.

Class	Description	Example

`AdabasConnection`	This class is used to create a main Adabas context. The parameter for `AdabasConnection` is an SQL-like string that contains all of the parameter needed to create a connection to Adabas. `AdabasSession` and `AdabasTarget` are created inside the `AdabasConnection` instance.
	
	`AdabasConnection conn = ↩`
	`AdabasConnection.createSession("ajc:map=MAPNAME;config=[1,4]");`
	`ReadRequest request = conn.createReadRequest(); ↩`
		ComplexQue

`AdabasSession`	The `AdabasSession` class defines the user and security credentials. If no credentials are required, the basic Adabas ID is automatically generated.
	
	`AdabasSession session = new ↩`
	`AdabasSession(AuthType.ADA_PW_SECURITY);`
	`session.addCredentials(username.getBytes(), password.getBytes());`
		AdabasAuth

`AdabasTarget`	The `AdabasTarget` class defines the Adabas target URL, and is the basic handle for Adabas access. If multiple request classes are used, the common handle underneath is the `AdabasTarget` instance managing synchronization.
	
	`AdabasTarget target = new AdabasTarget(dbid);`
	`target.open();`
	`target.et();`
		QueryWithM

Class	Description	Example

ReadRequest	The ReadRequest class handles a read of Adabas data. In the ReadRequest instance, a set of fields to be read is defined. The ReadRequest creates corresponding Adabas read definitions (metadata) in order to be able to call the query. In addition, the read offset, number of records and search criteria are provide to the ReadRequest instance to manage the corresponding Adabas read or search.
	
	`ReadRequest request = new ReadRequest("MAP");`
	`request.setStart(1);`
	`request.setLimit(10);`
	`request.readIsnSequence();`
		QueryWithMapping

StoreRequest	The StoreRequest class provides update and store functionality. A StoreRequest instance can define a set of fields, and these fields can be filled with value data. If a ReadRequest reads a record entry, this entry can be modified and updated in the database using the StoreRequest instance.
	
	`StoreRequest request = conn.createStoreRequest();`
	`RecordEntry entry = request.createRecordEntry(new String[] ↩`
	`{"NAME"});`
	`entry.addValue("NAME","ADAM");`
	`request.storeEntry(entry); ↩`
		StoreDataWithMapp

DeleteRequest	The DeleteRequest class is used to delete a single ISN or a set of ISNs.
	
	`DeleteRequest deleteRequest = new DeleteRequest(request);`
	`deleteRequest.setIsnList(isnList.toArray(new Long[0]));`
	`deleteRequest.deleteRecords(); ↩`
		DeleteExample.java

RecordDefinition	The RecordDefinition class describes the metadata information of the set of fields in a Request. The field information in the RecordDefinition can be overwritten, for example, the name or the field type can be changed.	QueryOverwriteMap
IRecordEntry	The IRecordEntry interface defines a current Adabas record used by a request. The request can read or write the record entry. The RecordEntry class contains the Adabas specific hierarchical representation such Adabas groups, period groups or multiple fields.	QueryUsingAdabasN
QueryResult	If a listener class is using the ReadRequest, the read will return a QueryResult class instance. The instance contains a set of query information. That contains information about number of records.	QueryUsingListener.
QueryResultList	The QueryResultList class provides the same information as the QueryResult class, but the QueryResultList contains a list of resulting records entries. The record list is stored in memory. When compared to QueryResult, QueryResultList is not recommended to be used with large lists of records.	ComplexQueryWithl
QueryResultCursor	QueryResultCursor is returned if readByCursor() read is used in a ReadRequest. The QueryResultCursor contains only a subset of records, unlike QueryResultList, which contains all of them.	

Class	Description	Example
`AdaFieldType`	The `AdaFieldType` class is part of the `RecordDefinition` class. It contains all field options and metadata used in the request. In addition, map definitions and field type definitions are stored in the class instance.	QueryOverw

`IAdaFieldValue`	The `IAdaFieldValue` interface provides data value information of the field.
	
	`IAdaFieldValue fieldValueFirstName = record.valueOf("NAME");`
	`fieldValueFirstName.toString()`
		QueryWithM

`IDataTypes.Types`	The Types enumeration contains all possible types of the fields. There is a default map automatism which generates, on behalf of the Adabas field or map definition, a corresponding field type to the metadata definition. It might be the case, that the default need to be overwritten.	AdabasToCs
`SearchTree`	In simple queries, the setSearch() methods of `ReadRequest` instances might be sufficient to do search calls. For complex search queries, the `SearchTree` defines and chains a number of search criteria to a complex search.	QueryAdaba

5 Adabas Client for Java and Eclipse

This tutorial provides an introduction to working with Adabas Client for Java and Eclipse.

Note: The Installation directory of Adabas Client for Java contains a set of JavaDocs for Adabas Client for Java, as well as example Java programs.

Adding the Adabas Client for Java Example Programs in Eclipse

This tutorial explains how to add the example programs from Adabas Client for Java into Eclipse. In the final step you will execute one of the example programs in Eclipse.

To add example programs from Adabas Client for Java in Eclipse

This tutorial assumes that a current version of Eclipse (currently Luna 4.4.1) is installed and running on your local machine. It also assumes that you have a demo Adabas database running on your local machine.

1 From the Eclipse **File** menu, choose **New** > **Java Project**.

The New Java Project dialog is displayed.

2 Enter the name of the new project (for example *AdabasClientForJavaExamples*) in the in the field **Project name**. Click on the button **Finish** to create the new project.

3 Select the new project in the Eclipse Package Explorer, then choose **Properties** from the Eclipse **File** menu.

Or:

Right-click on the new project in the Eclipse Package Explorer, then choose **Properties** from the context menu.

The Properties for *ProjectName* dialog is displayed.

4 Select **Java Build Path** on the left side of the dialog, then select the **Libraries** tab on the right side of the dialog.

The JARs and class folders on the build path are listed.

5 Click on the button **Add External JARs...** and then select the acj jar file from *install directory*\AdabasClientForJava\lib.

Note: The *install directory* (by default, this is "SoftwareAG") can be changed during the installation.

The JAR Selection dialog is displayed.

6 In the JAR Selection dialog, select the acj-<version>, log4j-<version>, slf4j-api-<version>, slf4j-log4j-<version> jar file from *install directory*\AdabasClientForJava\lib, then click on the button **Open**.

7 Click on the button **OK** in the Properties dialog to add the jar file to the Referenced Libraries.

The next step is to add the Javadoc location.

8 Expand the object acj jar on the right side of the Properties dialog and select **Javadoc location**, then click on the **Edit...** button.

The Javadoc for 'acj jar' dialog is displayed.

9 Select the radio button **Javadoc in archive**, then click on the **Browse...** button next to the field **Archive path**.

The Javadoc Archive Selection dialog is displayed.

10 In the Javadoc Archive Selection dialog, select the acj javadoc jar file from *install directory*\AdabasClientForJava\lib, then click on the button **Open** to add the Javadoc location.

Notes:

1. Add the value "docs" in the field `Path within Archive`.
2. Click on the button **Validate** for verification.
3. Click on the button **OK** in the dialogs Javadoc For 'acj jar' and Build Path to return to the Eclipse main screen.

The next step is to create a new Java package in the folder *src*.

11 Select the new project (in this case AdabasClientForJavaExamples) in the Eclipse Package Explorer, then click on the New Java Package icon in the tool bar.

Or:

Right-click on the new project, and then choose **New>Package** from the context menu.

The New Java Package dialog is displayed.

12 In the New Java Package dialog, enter the string *com.softwareag.adabas.query.examples.classic* in the field **Name**. Click on the button **Finish** to add the new package to the project.

Repeat this step, this time entering the string *com.softwareag.adabas.query.examples.mapping* in the field **Name**.

The next step is to import source files to the new packages.

13 Select the new package *com.softwareag.adabas.query.examples.classic* in the Eclipse Package Explorer, then choose **Import...** from the Eclipse **File** menu.

Or:

Right-click on the new package, and then choose **Import...** from the context menu.

The Import dialog is displayed.

14 In the Import dialog, select **General -> File System** and click on the button **Next**. In the subsequent Import dialog, click on the **Browse...** button next to the field **From directory** and navigate to the directory that contains the source files (in this case *install directory*\AdabasClientForJava\examples\classic). Click on the **Select All** button to select all of the source files to be imported. Click on the Finish button to import the source files to the new package.

Repeat this step for the new package *com.softwareag.adabas.query.examples.mapping*, this time importing the source files from *install directory*\AdabasClientForJava\examples\mapping).

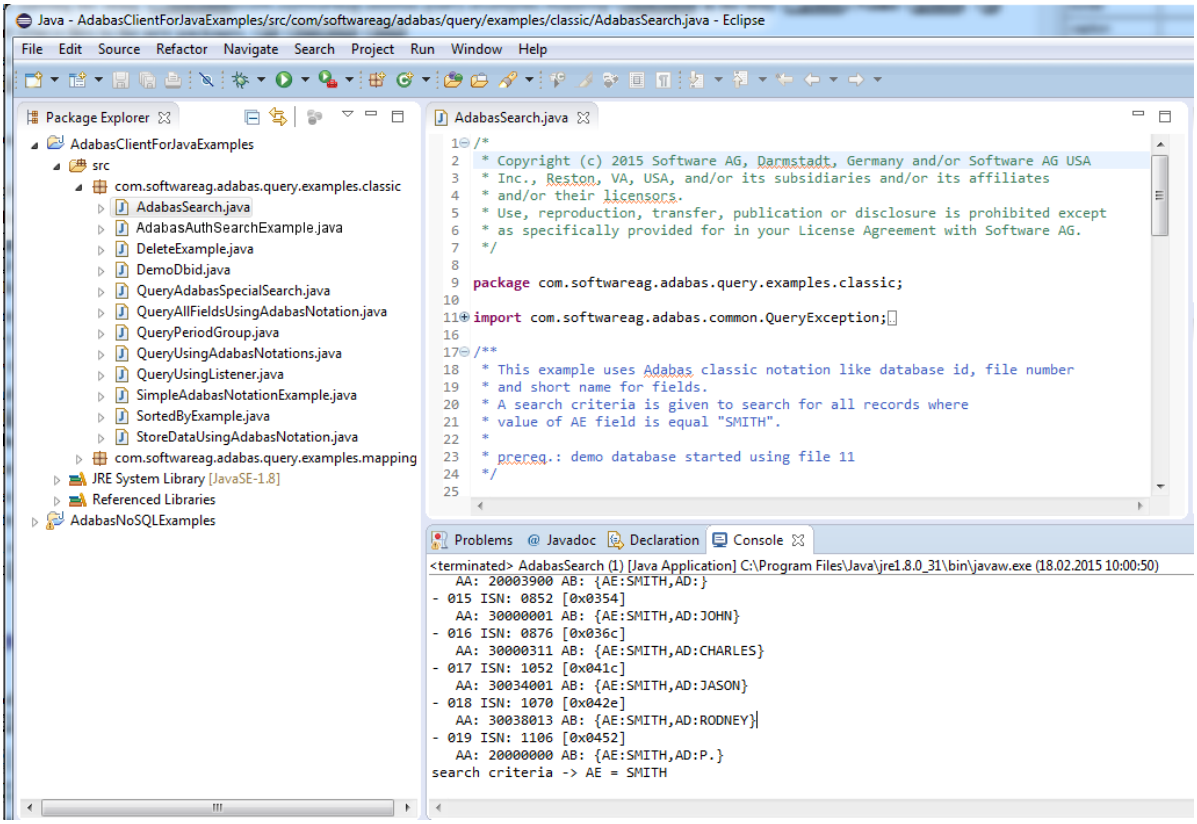
The next step is to verify that Javadoc is working correctly.

15 Double-click on the file *AdabasSearch.java* in the Eclipse Package Explorer to open it for editing. Move your cursor over an occurrence of the string `AdabasTarget` in the source code - the corresponding Javadoc text should be displayed in a popup window. Click on the icon **Open Attached Javadoc in Browser** to display the text in your web browser.

The final step is to execute one of the example programs.

16 Select the example program *AdabasSearch.java* in the Eclipse Package Explorer, then click on the run icon in the toolbar to run it; you will be prompted to enter the DBID of your demo database in the Eclipse Console window, then press Enter to run the program.

The results of running the program are displayed in the console of the Eclipse main screen, which will look something like this:



Running the Example Program AdabasAuthSearchExample

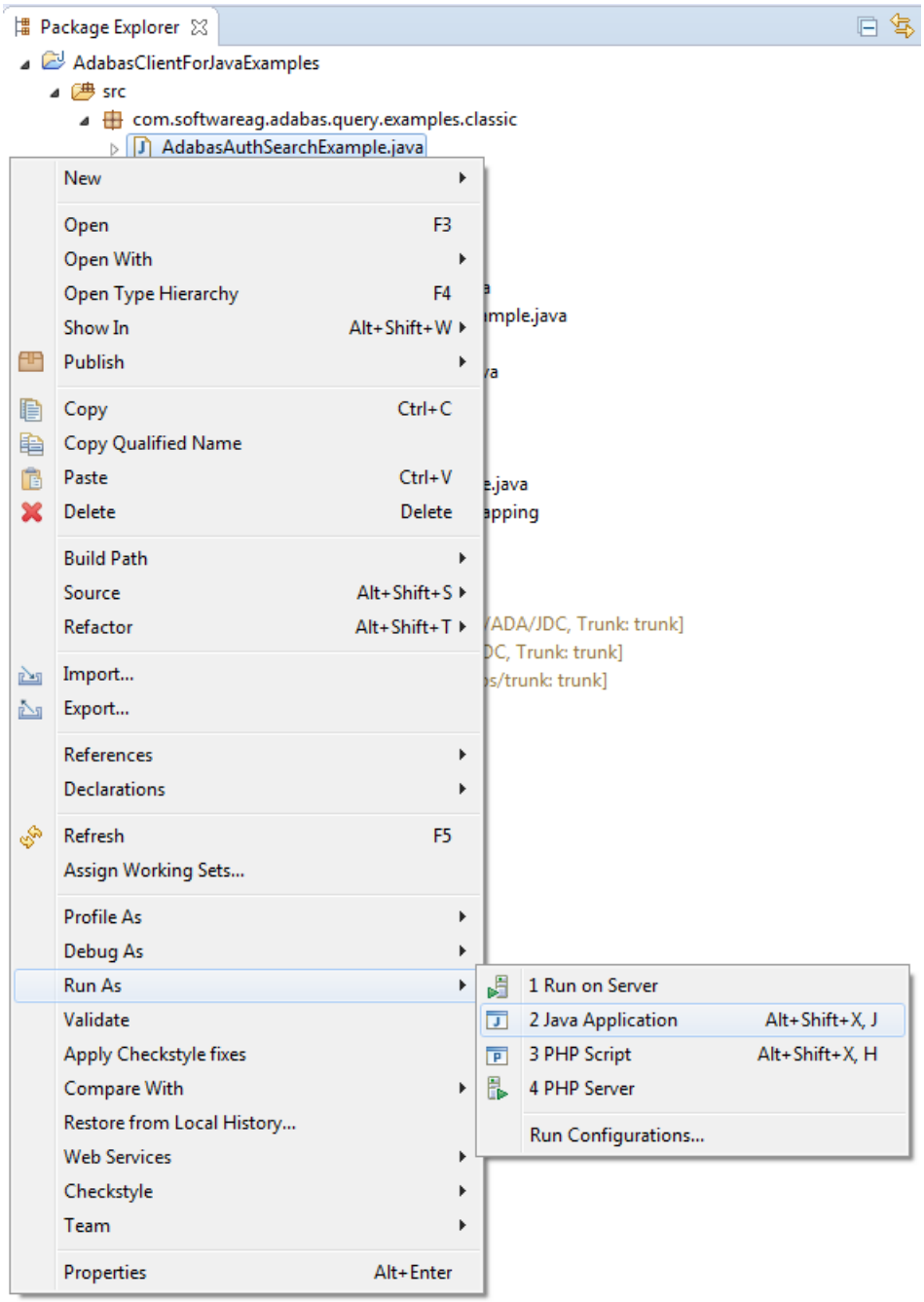
There are two ways in which the example program *AdabasAuthSearchExample* can be run from Eclipse:

- Use **Run As Java Application** - in this case, it is not possible to disable echoing of the password.
- Use **Show In Terminal** - in this case, it is possible to hide the password during input.

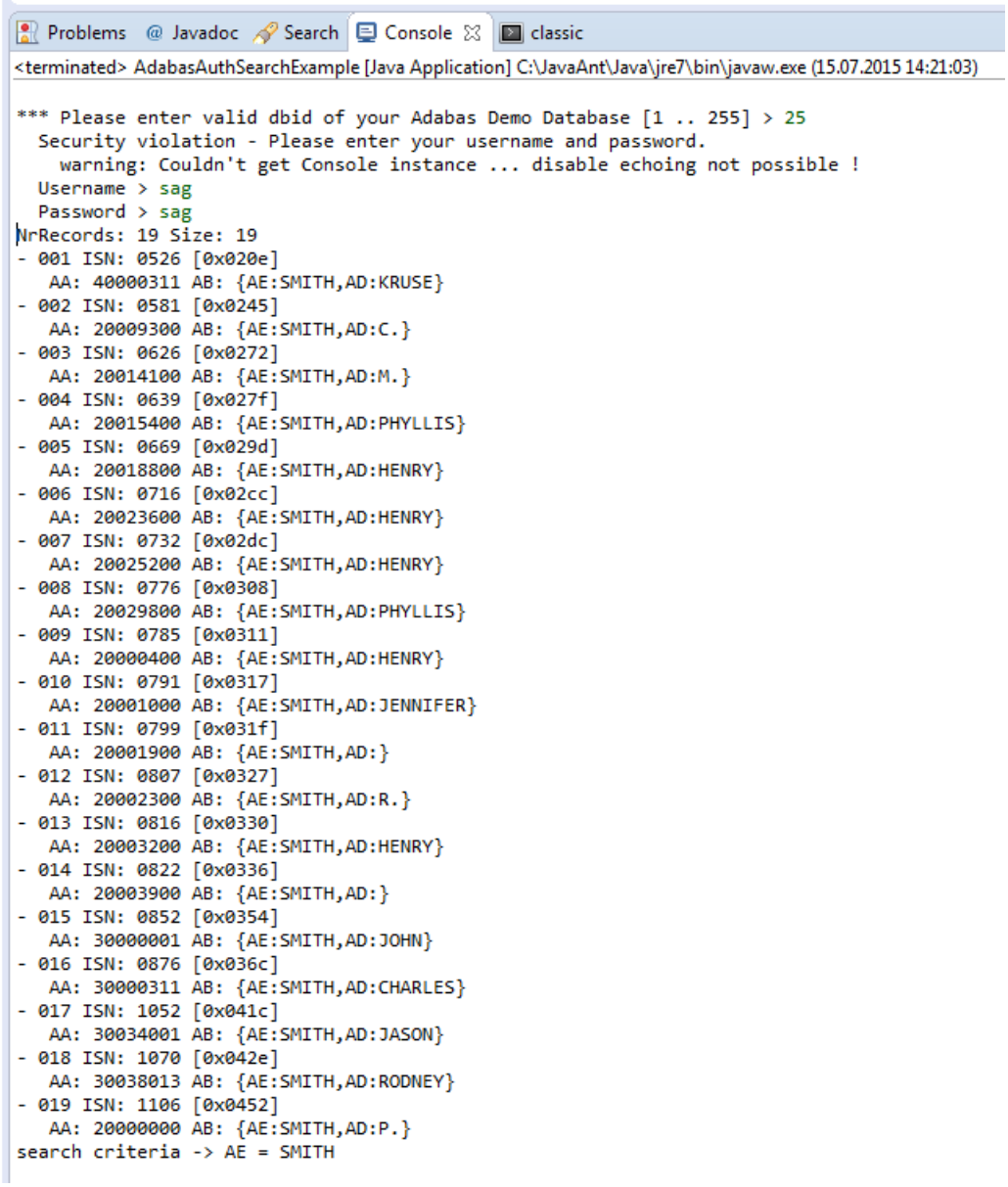
Examples of each case (including output) are shown below; in each case, you must first select the example program *AdabasAuthSearchExample.java* in the Eclipse Package Explorer.

Example: Run As Java Application

Click on **Run As** -> **Java Application** from the toolbar, as shown below:

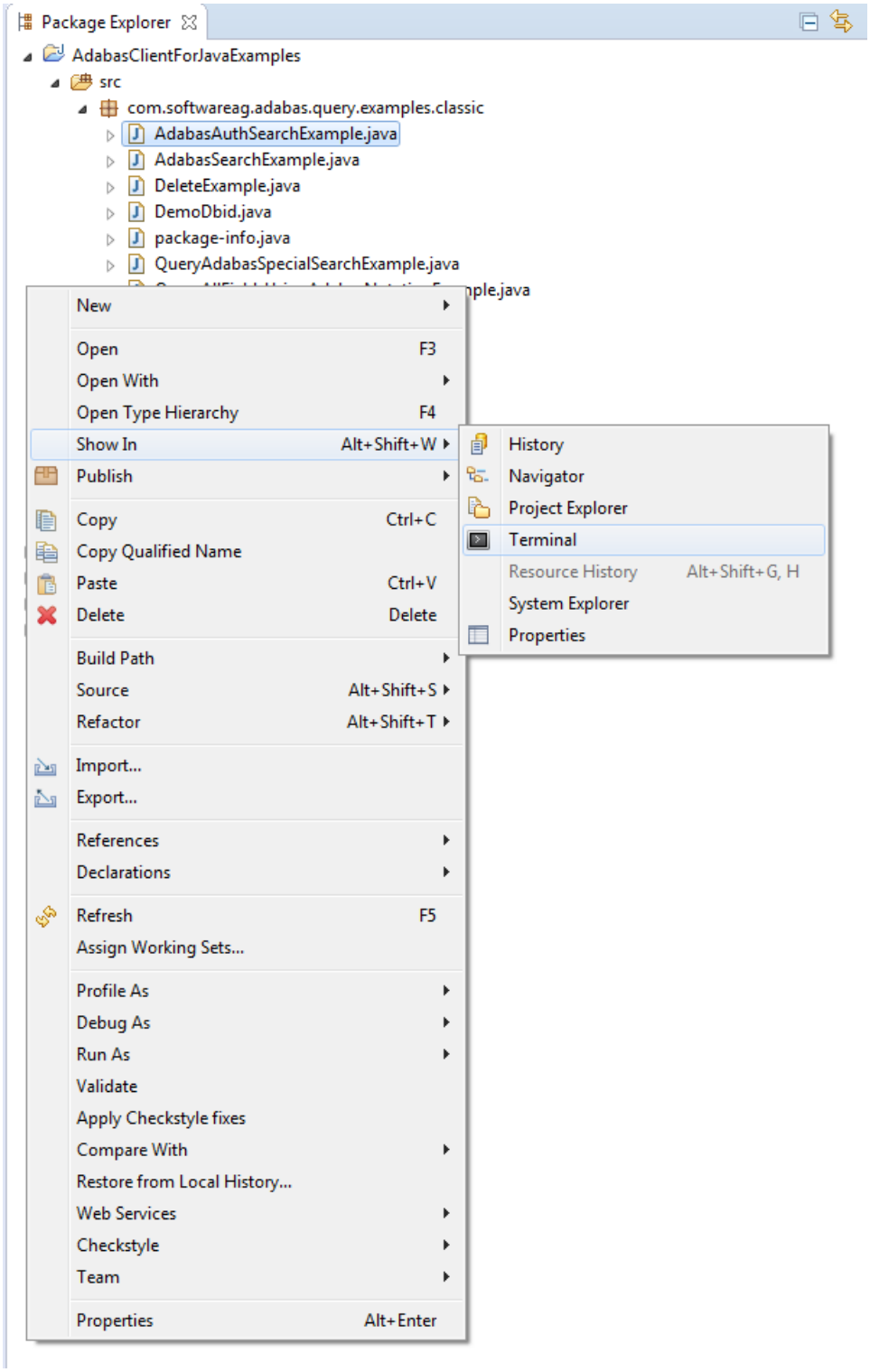


The output in the IDE console window will look something like this:

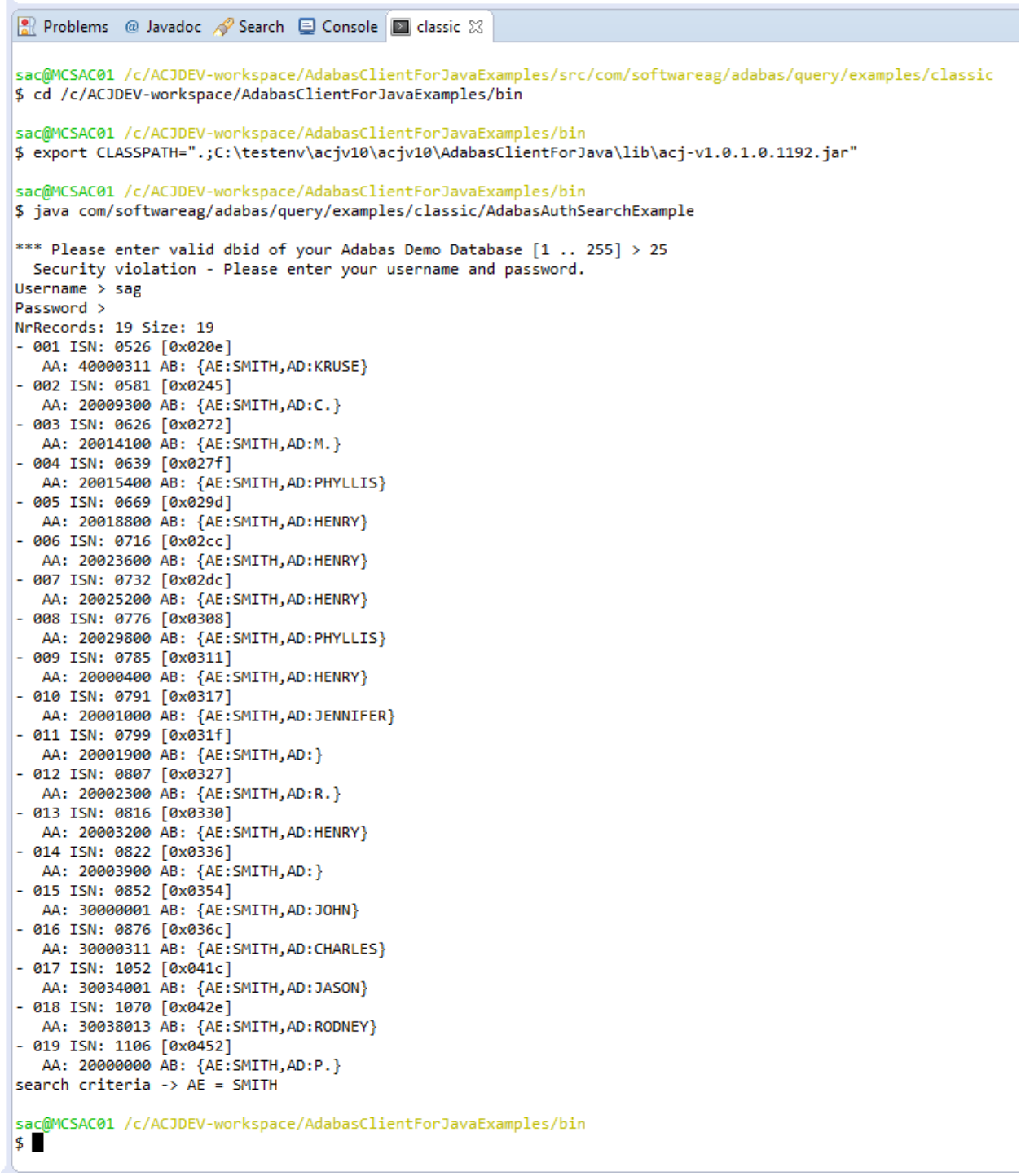


Example: Show In Terminal

Click on **Show In** -> **Terminal** from the toolbar, as shown below:



The output in the IDE classic Terminal window will look something like this:


Problems
Javadoc
Search
Console
classic
sac@MCSAC01 /c/ACJDEV-workspace/AdabasClientForJavaExamples/src/com/softwareag/adabas/query/examples/classic
$ cd /c/ACJDEV-workspace/AdabasClientForJavaExamples/bin
sac@MCSAC01 /c/ACJDEV-workspace/AdabasClientForJavaExamples/bin
$ export CLASSPATH=".;C:\testenv\acjv10\acjv10\AdabasClientForJava\lib\acj-v1.0.1.0.1192.jar"
sac@MCSAC01 /c/ACJDEV-workspace/AdabasClientForJavaExamples/bin
$ java com/softwareag/adabas/query/examples/classic/AdabasAuthSearchExample
*** Please enter valid dbid of your Adabas Demo Database [1 .. 255] > 25
Security violation - Please enter your username and password.
Username > sag
Password >
NrRecords: 19 Size: 19
- 001 ISN: 0526 [0x020e]
AA: 40000311 AB: {AE:SMITH,AD:KRUSE}
- 002 ISN: 0581 [0x0245]
AA: 20009300 AB: {AE:SMITH,AD:C.}
- 003 ISN: 0626 [0x0272]
AA: 20014100 AB: {AE:SMITH,AD:M.}
- 004 ISN: 0639 [0x027f]
AA: 20015400 AB: {AE:SMITH,AD:PHYLLIS}
- 005 ISN: 0669 [0x029d]
AA: 20018800 AB: {AE:SMITH,AD:HENRY}
- 006 ISN: 0716 [0x02cc]
AA: 20023600 AB: {AE:SMITH,AD:HENRY}
- 007 ISN: 0732 [0x02dc]
AA: 20025200 AB: {AE:SMITH,AD:HENRY}
- 008 ISN: 0776 [0x0308]
AA: 20029800 AB: {AE:SMITH,AD:PHYLLIS}
- 009 ISN: 0785 [0x0311]
AA: 20000400 AB: {AE:SMITH,AD:HENRY}
- 010 ISN: 0791 [0x0317]
AA: 20001000 AB: {AE:SMITH,AD:JENNIFER}
- 011 ISN: 0799 [0x031f]
AA: 20001900 AB: {AE:SMITH,AD:}
- 012 ISN: 0807 [0x0327]
AA: 20002300 AB: {AE:SMITH,AD:R.}
- 013 ISN: 0816 [0x0330]
AA: 20003200 AB: {AE:SMITH,AD:HENRY}
- 014 ISN: 0822 [0x0336]
AA: 20003900 AB: {AE:SMITH,AD:}
- 015 ISN: 0852 [0x0354]
AA: 30000001 AB: {AE:SMITH,AD:JOHN}
- 016 ISN: 0876 [0x036c]
AA: 30000311 AB: {AE:SMITH,AD:CHARLES}
- 017 ISN: 1052 [0x041c]
AA: 30034001 AB: {AE:SMITH,AD:JASON}
- 018 ISN: 1070 [0x042e]
AA: 30038013 AB: {AE:SMITH,AD:RODNEY}
- 019 ISN: 1106 [0x0452]
AA: 20000000 AB: {AE:SMITH,AD:P.}
search criteria -> AE = SMITH
sac@MCSAC01 /c/ACJDEV-workspace/AdabasClientForJavaExamples/bin
$