JADE Audit Access

Version 2018

Contents

JADE Audit Access

Introduction

As JADE journals are designed for optimized performance and therefore not human-readable, the **JadeAuditAccess** class is provided as a tool for the extraction of useful information from JADE journals.

Note This module assumes you have completed the JADE Developer's course and therefore have a copy of the **Banking** system and that you have used it, generating journals. If this is not the case, simply download it from https://github.com/jadesoftwarenz/JADE-Banking-Schema.

JADE Journals

Journals provide several functions necessary to the smooth and consistent running of a JADE database, including:

- Recovering the database to a consistent state after a crash.
- Undoing the effects of aborted transactions.
- Rolling-forwards after a restore from backup.
- Maintaining a redundant duplicate of the database when using the Synchronized Database Service (SDS).

As there are important performance concerns with journaling every database action, it is not safe to access the current journal, and journals are stored on disk in a manner that is not only not human-readable, but it is not in any documented format. It is therefore not recommended that you try to read the journal files directly.

These journals, however, contain a wealth of useful information, including:

Information	Description
Audit	Which properties of which objects have changed, and what are their before and after states?
Data lifetimes	How often class properties change. Which properties change often, and which are mostly static?
Diagnostics	What exactly was the scope of a specific transaction; that is, what data did it change?
Extraction	The objects that have been modified in a specified period.
Performance	How long did transactions take to process, and how many are being processed?
Security	Who initiated specific transactions or any transaction that changed a specific object or property?

The journals can also be useful for the following.

Benefit	Description
Independence	Journals are a source that can be kept separate from the database, and therefore can provide access to information about the database without relying on application code.
Understanding	The information provided in journals can provide insight into data relationships and business procedures.

As such, there is benefit in the ability to extract this useful information out of the journals and present it in a human-readable format. The **JadeAuditAccess** class of **RootSchema** provides this ability.

Description Files

To convert the journal files to a human-readable format, a Database Description File (DDF) is used to interpret the structure of the class objects.

Whenever the format of a persistent object changes (for example, adding or removing a property of a class with at least one instantiated object), the structure of the journal changes. These changes also require a reorganization of the database itself, so an easy way to generate the required description files is to add the following parameter to your JADE initialization file.

```
[PersistentDb]
UseJournalDescriptions=true
```

Specifying this parameter with a value of **true** causes a new description file to automatically be generated after every reorganization.

You may sometimes need to generate a description file manually. For example, creating a new class in the database changes the journal structure but does not require a reorganization and therefore does not automatically generate a new description file. You can also generate an initial description file if the **UseJournalDescriptions** parameter has been set to **false**, by using the **JadeAuditAccess** class **generateDescription** Method. This method requires no parameters and generates a description file in the default location (that is, the **journals** folder within the **system** directory of your JADE installation).

```
makeAuditDescription();

vars
    auditAccess : JadeAuditAccess;
begin
    create auditAccess transient;

    beginTransaction;
    auditAccess.generateDescription();
    commitTransaction;
epilog
    delete auditAccess;
end;
```

Note As the **JadeAuditAccess** class **generateDescription** method updates the database and is audited, it therefore must be performed inside a transaction state.

Using the JadeAuditAccess Class to Read Journals

The main steps to read journals with Audit Access functionality are as follows.

1. Locate the first journal
2. Move between journals safely
3. Parse information out of each journal

The challenge with locating the first journal is that it cannot be guaranteed that the journals start at **1**, as they typically start in the range **20** through **23**, depending on the version of JADE you are using. However, in most cases, only a subset of journals is audited and, in this case, the first journal to be audited can simply be passed in the **pJournalNumber** parameter of the **JadeAuditAccess** class **getJournal** method.

The **getJournal** method parameters are as follows.

- **pDirectory**, which is a **String** value specifying the directory containing the journals. This can be manually specified, or it can be obtained from the **getCurrentJournalDirectory** method of the **JadeDatabaseAdmin** class.
- **pJournalNumber**, which is an **Integer** value specifying the number of the journal to open.
- **pRecordOffset**, which is an **Integer IO** value that is set to the position in the file after the header is skipped.

The **getJournal** method returns zero (**0**) if the journal is available, or exception 3125 *(A required transaction journal was not found)* is raised if it is not found.

Moving between journals is easy, as the **JadeAuditAccess** class **getNextJournal** method retrieves the next journal, although it *does* depend on a journal already being open (that is, the **getJournal** method must be called at least once).

The **getNextJournal** method returns zero (**0**) if an available next journal exists, or raises exception 3036 (*The database file being opened is required but was not found*) if there is no next journal. The most common strategy for this case is to assume that there will be a next journal, and simply handle the exception, if not.

When a journal is opened, the **JadeAuditAccess** class **getNextRecord** method can be used. This method requires several **output** parameters, and it update these parameters with the data from the journal as it is parsed.

Note **output** parameters are used to pass information back from the called method to the calling method, and as such, should not be initialized in the calling method. For more details, see "Parameters", in Chapter 1 of the *JADE Developer's Reference*.

The parameters (which are all **output** parameters) of the **getNextRecord** method are as follows.

Name	Type	Description
pType	Integer	Represents the type of action recorded in the entry, using the **JadeAuditAccess** class constants; for example, **51** is **Jaa_Type_BeginTransaction**.
pObjectType	Integer	Represents the type of object recorded in the entry, using the **JadeAuditAccess** class constants; for example, **9** is **Jaa_Object_Collection**.
pRecordOffset	Integer	Current position in the journal file.

Name	Type	Description
pTimestamp	TimeStamp	TimeStamp of when the entry was generated.
pSerialNumber	Decimal	Audit serial number of the entry.
pTransactionId	Decimal	Unique identifier for the transaction.
pOid	String	Object identifier (oid) of the object in the entry.
pClassNumber	Integer	Class number of the object in the entry.
pEdition	Integer	Update count of the object in the entry.

Exercise 1 – Creating a Description File

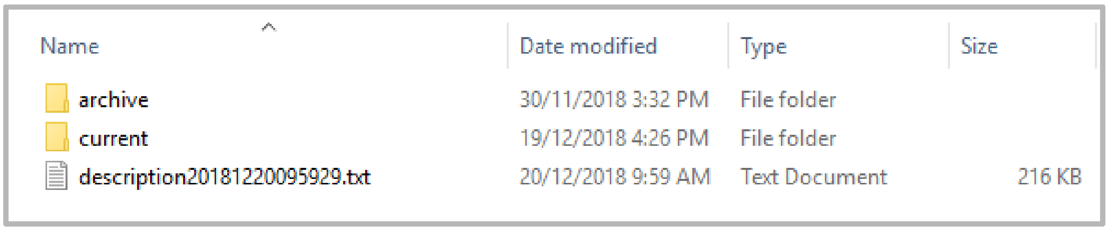
In this exercise, you will manually create a description file for your database's journals and set the JADE initialization file (**jade.ini**, by default) parameter required to have it automatically create description files on reorganization.

1. Create a schema called **AuditSchema** and open it in the Class Browser.
2. Add a JadeScript method called **createAuditDescription** and code it as follows. (Use the F4 shortcut key to find **JadeScript** in the Class Browser.)

```
createAuditDescription();

vars
    auditAccess : JadeAuditAccess;
begin
    create auditAccess transient;

    beginTransaction;
    auditAccess.generateDescription();
    commitTransaction;

epilog
    delete auditAccess;
end;
```

3. Run the method and then navigate to the journals directory (***install-dir*\system\journals**, where ***install-dir*** is the location of your JADE installation).

You should see that a description file like the following has been generated.



Name	Date modified	Type	Size
archive	30/11/2018 3:32 PM	File folder	
current	19/12/2018 4:26 PM	File folder	
description20181220095929.txt	20/12/2018 9:59 AM	Text Document	216 KB

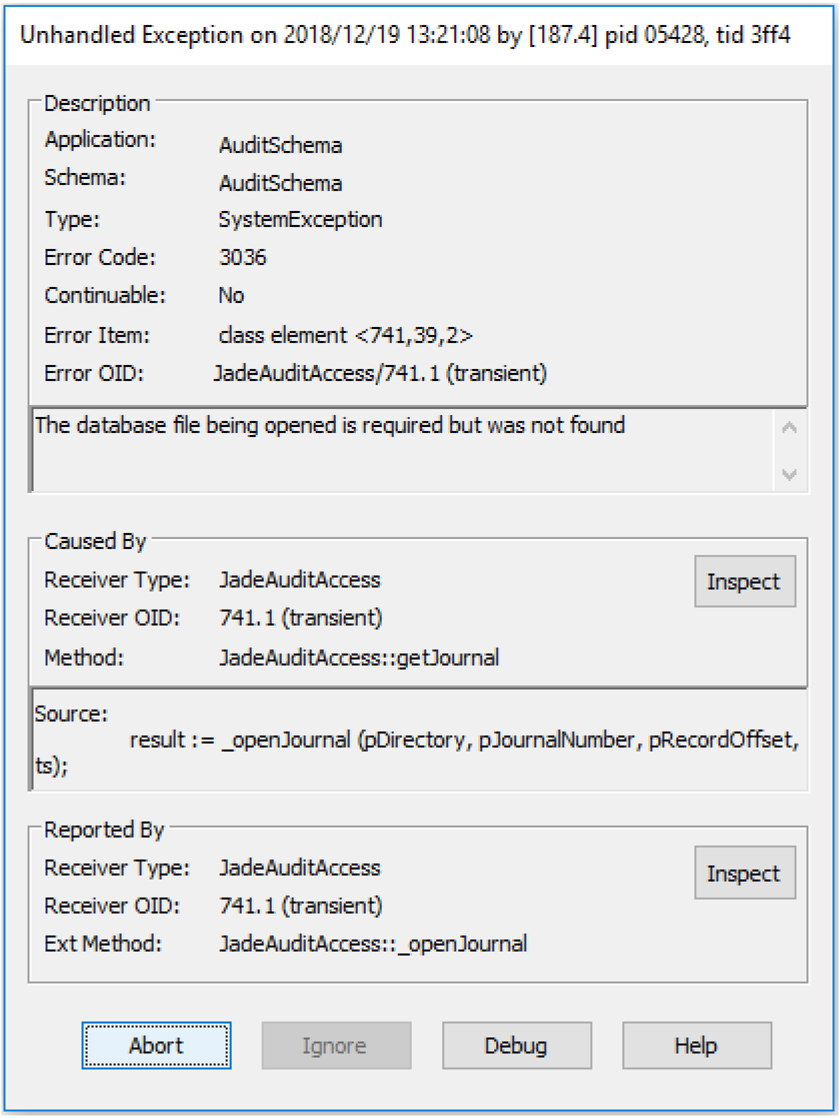
Exercise 2 – Opening a Journal

In this exercise, you will use the **JadeAuditAccess** class to open a journal and then retrieve various information about the journal.

1. Add a JadeScript method called **openJournal** and code it as follows.

```
openJournal();

constants
    FirstJournal = 1;
vars
    auditAccess  : JadeAuditAccess;
    dbAdmin      : JadeDatabaseAdmin;
    offset       : Integer;
begin
    create auditAccess transient;
    create dbAdmin transient;

    auditAccess.getJournal(dbAdmin.getCurrentJournalDirectory, FirstJournal, offset);

    write auditAccess.getJournalName;
    write auditAccess.getJournalNumber;
    write auditAccess.getJournalPath;

epilog
    delete auditAccess;
    delete dbAdmin;
end;
```

2. Run the method, which should raise the following exception.



Unhandled Exception on 2018/12/19 13:21:08 by [187.4] pid 05428, tid 3ff4

Description

Application:	AuditSchema
Schema:	AuditSchema
Type:	SystemException
Error Code:	3036
Continuable:	No
Error Item:	class element <741,39,2>
Error OID:	JadeAuditAccess/741.1 (transient)

The database file being opened is required but was not found

Caused By

Receiver Type:	JadeAuditAccess
Receiver OID:	741.1 (transient)
Method:	JadeAuditAccess::getJournal

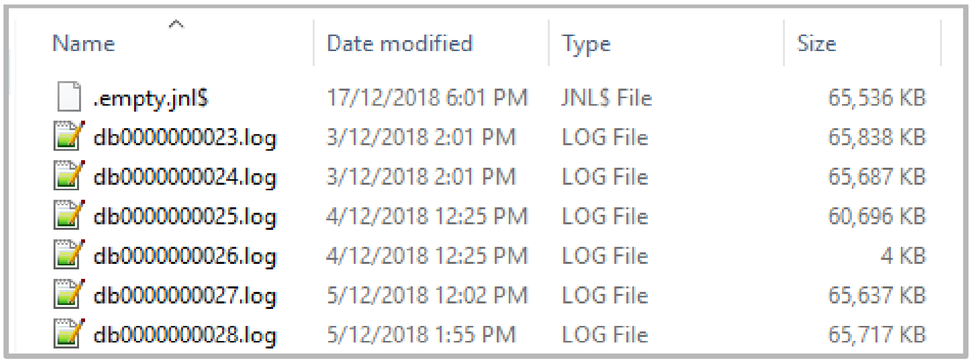
```
Source:
        result := _openJournal (pDirectory, pJournalNumber, pRecordOffset,
ts);
```

Reported By

Receiver Type:	JadeAuditAccess
Receiver OID:	741.1 (transient)
Ext Method:	JadeAuditAccess::_openJournal

3. Navigate to your journal directory in your file system (***install-dir*/system/journals/current**).

4. Change the **FirstJournal** constant value in your **openJournal** method to the first journal in the directory. (In the following example, the first journal number is 23.)



Name	Date modified	Type	Size
.empty.jnl$	17/12/2018 6:01 PM	JNL$ File	65,536 KB
db0000000023.log	3/12/2018 2:01 PM	LOG File	65,838 KB
db0000000024.log	3/12/2018 2:01 PM	LOG File	65,687 KB
db0000000025.log	4/12/2018 12:25 PM	LOG File	60,696 KB
db0000000026.log	4/12/2018 12:25 PM	LOG File	4 KB
db0000000027.log	5/12/2018 12:02 PM	LOG File	65,637 KB
db0000000028.log	5/12/2018 1:55 PM	LOG File	65,717 KB

5. Run the method again. The journal name, number, and location should be written to the Jade Interpreter Output Viewer.

Exercise 3 – Reading an Entry from a Journal

In this exercise, you will extend the **openJournal** method to read the first entry from the first journal.

1. Add the following to the variable declaration (**vars**) section of the **openJournal** method.

```
openJournal();

constants
    FirstJournal = 23;
vars
    auditAccess  : JadeAuditAccess;
    dbAdmin      : JadeDatabaseAdmin;
    offset       : Integer;

    // output parameters for getNextRecord
    entryType           : Integer;
    entryObjectType     : Integer;
    entryTimeStamp      : TimeStamp;
    entrySerialNumber   : Decimal[12];
    entryTransactionID  : Decimal[12];
    entryOID            : String;
    entryClassNumber    : Integer;
    entryEdition        : Integer;
```

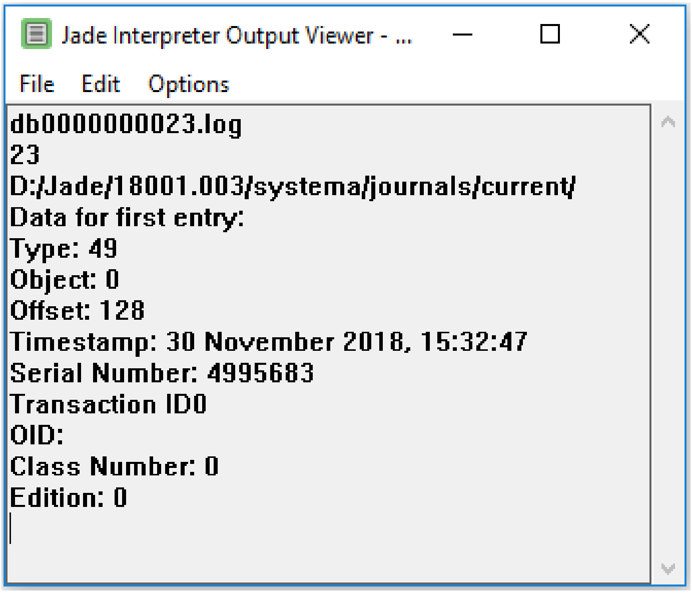
Tip When calling a method that requires a lot of parameters, it can be a good idea to put each parameter on a separate line to aid readability.

JADE uses semicolons to detect the end of a statement, so you can use line breaks as needed.

2. Modify the **openJournal** method, as follows.

```
begin
    create auditAccess transient;
    create dbAdmin transient;

    auditAccess.getJournal(dbAdmin.getCurrentJournalDirectory, FirstJournal, offset);

    write auditAccess.getJournalName;
    write auditAccess.getJournalNumber;
    write auditAccess.getJournalPath;

    auditAccess.getNextRecord(
                        entryType,
                        entryObjectType,
                        offset,
                        entryTimeStamp,
                        entrySerialNumber,
                        entryTransactionID,
                        entryOID,
                        entryClassNumber,
                        entryEdition
                        );

    write "Data for first entry:";
    write "Type: " & entryType.String;
    write "Object: "& entryObjectType.String;
    write "Offset: " & offset.String;
    write "Timestamp: " & entryTimeStamp.String;
    write "Serial Number: " & entrySerialNumber.String;
    write "Transaction ID" & entryTransactionID.String;
    write "OID: " & entryOID;
    write "Class Number: " & entryClassNumber.String;
    write "Edition: " & entryEdition.String;

epilog
    delete auditAccess;
    delete dbAdmin;
end;
```

3. Run the method. The Jade Interpreter Output Viewer should then look like the following.



```
db0000000023.log
23
D:/Jade/18001.003/systema/journals/current/
Data for first entry:
Type: 49
Object: 0
Offset: 128
Timestamp: 30 November 2018, 15:32:47
Serial Number: 4995683
Transaction ID0
OID:
Class Number: 0
Edition: 0
```

Consulting the list of **JadeAuditAccess** class constants, we see that a Type of **49** is a **Database Open**. As such, there is no associated object.

Note For details, see the **JadeAuditAccess** class in the *JADE Encyclopaedia of Classes (Volume 1)*, available at https://www.jadeworld.com/docs/jade-2018/Default.htm.

Exercise 4 – Reading an Entire Journal

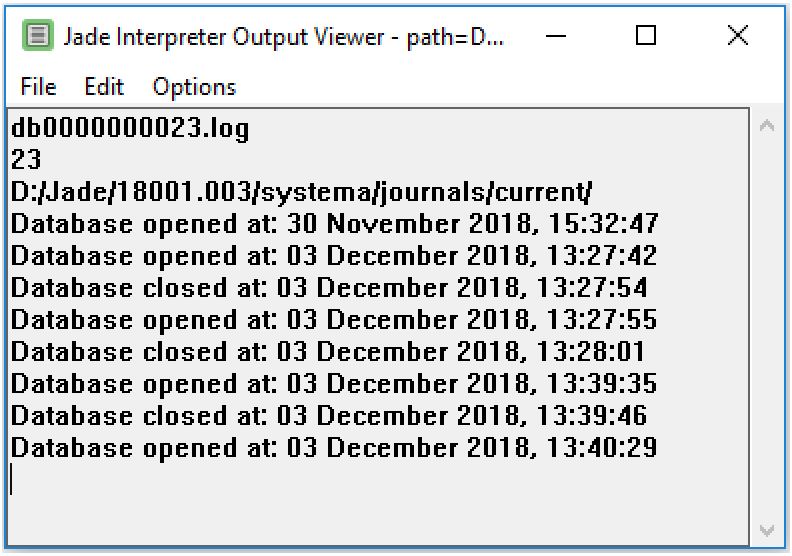
In this exercise, you will read an entire journal, displaying only the entries that relate to a database open or close action.

1. Modify the **openJournal** method as follows.

```
begin
    create auditAccess transient;
    create dbAdmin transient;

    auditAccess.getJournal(dbAdmin.getCurrentJournalDirectory, FirstJournal, offset);

    write auditAccess.getJournalName;
    write auditAccess.getJournalNumber;
    write auditAccess.getJournalPath;

    while auditAccess.getNextRecord(
                        entryType,
                        entryObjectType,
                        offset,
                        entryTimeStamp,
                        entrySerialNumber,
                        entryTransactionID,
                        entryOID,
                        entryClassNumber,
                        entryEdition
                        )do
        if entryType = auditAccess.Jaa_Type_DatabaseOpen then
            write "Database opened at: " & entryTimeStamp.String;
        elseif entryType = auditAccess.Jaa_Type_DatabaseClose then
            write "Database closed at: " & entryTimeStamp.String;
        endif;
    endwhile;
epilog
    delete auditAccess;
    delete dbAdmin;
end;
```

2. Run the method.

   The Jade Interpreter Output Viewer should then look like the following.



```
db0000000023.log
23
D:/Jade/18001.003/systema/journals/current/
Database opened at: 30 November 2018, 15:32:47
Database opened at: 03 December 2018, 13:27:42
Database closed at: 03 December 2018, 13:27:54
Database opened at: 03 December 2018, 13:27:55
Database closed at: 03 December 2018, 13:28:01
Database opened at: 03 December 2018, 13:39:35
Database closed at: 03 December 2018, 13:39:46
Database opened at: 03 December 2018, 13:40:29
```

3. Position the caret on the **getNextRecord** method and press the F1 shortcut key, to display the documentation about that method and use it to answer the following questions.

   a. What is the difference between the **getNextRecord** and **getNextRecordUTC** methods?

   b. Why does it work having the **getNextRecord** method as the condition for a **while** loop?

   Tip You may need to read the documentation for the **getNextRecordUTC** method, for the answer.

Exercise 5 – Reading All Journals of a Database

In this exercise, you will extend the **openJournal** method to read all journals of the database rather than just the first one.

1. Modify the **openJournal** method as follows.

```
begin
    create auditAccess transient;
    create dbAdmin transient;

    auditAccess.getJournal(dbAdmin.getCurrentJournalDirectory, FirstJournal, offset);

    while true do
        write auditAccess.getJournalName;
        write auditAccess.getJournalNumber;
        write auditAccess.getJournalPath;

        while auditAccess.getNextRecord(
                            entryType,
                            entryObjectType,
                            offset,
                            entryTimeStamp,
                            entrySerialNumber,
                            entryTransactionID,
                            entryOID,
                            entryClassNumber,
                            entryEdition
                            )do
            if entryType = auditAccess.Jaa_Type_DatabaseOpen then
                write "Database opened at: " & entryTimeStamp.String;
            elseif entryType = auditAccess.Jaa_Type_DatabaseClose then
                write "Database closed at: " & entryTimeStamp.String;
            endif;
        endwhile;
        if not auditAccess.getNextJournal() = 0 then
            break;
        endif;
    endwhile;
epilog
    delete auditAccess;
    delete dbAdmin;
end;
```

2. Run the method.

   You will notice that it will successfully open each journal, one by one, and display any database open and close actions. However, when it reaches the end, an exception is raised.

3. To handle this exception, create a JadeScript method called **openJournalExceptionHandler** and code it as follows.

```
openJournalExceptionHandler(e: Exception; gotJournal: Boolean io): Integer;

begin
    if e.errorCode = JErr_DbFileNotFound then
        gotJournal:= false;
        return Ex_Resume_Next;
    else
        return Ex_Pass_Back;
    endif;
end;
```

4. Add a new **Boolean** called **gotJournal** to the **vars** list, as follows.

```
openJournal();

constants
    FirstJournal = 23;
vars
    auditAccess   : JadeAuditAccess;
    dbAdmin       : JadeDatabaseAdmin;
    offset        : Integer;

    // output parameters for getNextRecord
    entryType            : Integer;
    entryObjectType      : Integer;
    entryTimeStamp       : TimeStamp;
    entrySerialNumber    : Decimal[12];
    entryTransactionID   : Decimal[12];
    entryOID             : String;
    entryClassNumber     : Integer;
    entryEdition         : Integer;

    gotJournal           : Boolean;
```

5. Initialize it to **true**, as follows.

```
begin
    gotJournal := true;

    create auditAccess transient;
    create dbAdmin transient;
```

6. Instead of using a *while true* loop, we can now use while **gotJournal**, relying on it being set to **false** by the exception handler when there are no more journals.

Modify **openJournal**, as follows.

```
begin
    gotJournal := true;

    create auditAccess transient;
    create dbAdmin transient;

    auditAccess.getJournal(dbAdmin.getCurrentJournalDirectory, FirstJournal, offset);

    on Exception do openJournalExceptionHandler(exception, gotJournal);
    while gotJournal do
        write auditAccess.getJournalName;
        write auditAccess.getJournalNumber;
        write auditAccess.getJournalPath;

        while auditAccess.getNextRecord(
                                    entryType,
                                    entryObjectType,
                                    offset,
                                    entryTimeStamp,
                                    entrySerialNumber,
                                    entryTransactionID,
                                    entryOID,
                                    entryClassNumber,
                                    entryEdition
                                    )do
            if entryType = auditAccess.Jaa_Type_DatabaseOpen then
                write "Database opened at: " & entryTimeStamp.String;
            elseif entryType = auditAccess.Jaa_Type_DatabaseClose then
                write "Database closed at: " & entryTimeStamp.String;
            endif;
        endwhile;
        auditAccess.getNextJournal();
    endwhile;
epilog
    delete auditAccess;
    delete dbAdmin;
end;
```

7. Run the method. It should no longer raise an exception when finishing.

Bonus Exercise – Finding the First Journal

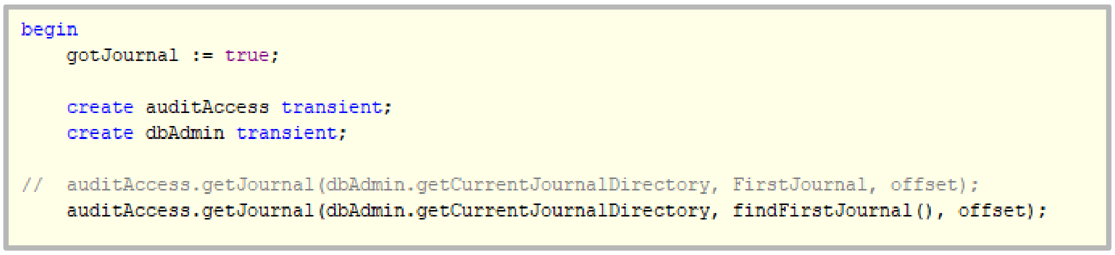
In the previous exercises, we hard-coded the number of the first journal.

In this bonus exercise, you will attempt to find the first journal in a general way. This exercise is optional and as such, less instruction will be given than in normal exercises.

1. Create a JadeScript method called **findFirstJournal** and code it as follows (for now).

```
findFirstJournal() : Integer;

vars

begin
    return 23;
end;
```

2. Modify the **openJournal** method as follows.



```
begin
    gotJournal := true;

    create auditAccess transient;
    create dbAdmin transient;

//  auditAccess.getJournal(dbAdmin.getCurrentJournalDirectory, FirstJournal, offset);
    auditAccess.getJournal(dbAdmin.getCurrentJournalDirectory, findFirstJournal(), offset);
```

This image is the first portion only of the method.

3. Your challenge is now to find an implementation of the **findFirstJournal** method that finds and returns the number of the first journal (which can vary from system to system).

Tips

You could:

- Try calling the **JadeAuditAccess** class **getJournal** method in a loop, trying journal numbers and handling the exceptions on misses
- Use the **FileFolder** class **files** method to look at the files in the default journal directory

Journal Filtering

Journal filtering is a powerful tool that extracts only the relevant data out of a set of journals.

The **JadeAuditAccess** class provides for several methods that enable you to set up filters that will then skip non-matching results when calling the **getNextRecord** method. These methods include the following.

- **registerFilterClass**, which takes a class number as a parameter and causes the **getNextRecord** method to return only events relating to instances of the specified class number.
- **registerFilterClassName**, which takes a schema name and a class name as parameters and causes the **getNextRecord** method to return only events relating to instances of the specified class within the specified schema.

  As this method relies on the journal's description file, it must be called after the **getJournal** method is successfully called.
- **registerFilterCollection**, which takes the number of a collection class and the number of a parent class.

  If the parent class number is set to zero (**0**), any event relating to the specified collection class is returned by the **getNextRecord** method; otherwise, only events relating to the specified collection of the specified parent class are returned.
- **registerFilterCollectionName**, which takes three parameters: a schema name, a class name, and a reference property name.

  It causes only those events that relate to the collection specified in the reference property of the specified class of the specified schema to be returned by the **getNextRecord** method.

  As this method relies on the journal's description file, it must be called after the **getJournal** method is successfully called.
- **registerFilterTimeRange**, which takes two timestamp parameters: one for a start time and one for an end time. Only the events between the specified start time and end time are returned by the **getNextRecord** method.

If the start time or end time parameter is not required, a **null** value accepts all events before or after the time, respectively.

If the start time is to be used (that is, it is set to a non-null value), the **registerFilterTimeRange** method must be called *before* the **getJournal** method.

- **registerFilterTimeRangeUTC**, which behaves the same as the **registerFilterTimeRange** method except that the timestamps are in Coordinated Universal Time (UTC).

Any of these methods other than the time range methods can be reversed by calling the **JadeAuditAccess** class **setFilterExcludes** method.

The **setFilterExcludes** method takes a **Boolean** value as a parameter, and if passed **true**, changes the behavior of the filter to cause the **getNextRecord** method to return everything except the filtered events rather than only the filtered events. This can be reversed back to the normal behavior by calling it again, passing it a **false** value.

Note When filtering by class or collection, a filter must be set for both collection and class.

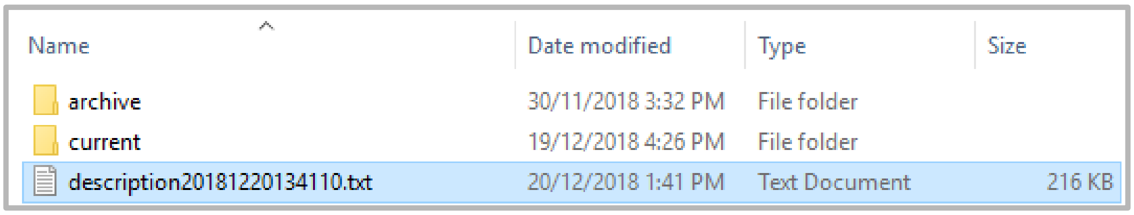
To filter out all non-collection classes or all collections, simply use a **RootSchema** class such as the **Schema** class (that is, class number **1**).

Exercise 6 – Filtering by Class

In this exercise, you will create a filter to show all events relating to the **Customer** class of **BankingModelSchema** that have been logged.

1. Navigate to your journal directory to find the timestamp of your most-recent description file (in this example, **20181220134110**). This number is based on the time and date when the description was generated.

   The following image shows one generated on December 12th, 2018 at 13:41:10).



Name	Date modified	Type	Size
archive	30/11/2018 3:32 PM	File folder	
current	19/12/2018 4:26 PM	File folder	
description20181220134110.txt	20/12/2018 1:41 PM	Text Document	216 KB

2. In **AuditSchema**, create a JadeScript method called **filterByCustomer** and code it as follows, replacing **<timestamp>** in the **loadDescriptionByName** call with the located timestamp.

```
filterByCustomer();

constants
    FirstJournal = 23;

vars
    auditAccess : JadeAuditAccess;
    dbAdmin     : JadeDatabaseAdmin;
    offset      : Integer;
    gotJournal  : Boolean;
    // output parameters for getNextRecord
    entryType           : Integer;
    entryObjectType     : Integer;
    entryTimeStamp      : TimeStamp;
    entrySerialNumber   : Decimal[12];
    entryTransactionID  : Decimal[12];
    entryOID            : String;
    entryClassNumber    : Integer;
    entryEdition        : Integer;
begin
    gotJournal := true;

    create auditAccess transient;
    create dbAdmin transient;

    auditAccess.getJournal(dbAdmin.getCurrentJournalDirectory, FirstJournal, offset);
    auditAccess.loadDescriptionByName("description<timestamp>.txt");
    auditAccess.registerFilterClassName("BankingModelSchema", "Customer");
    auditAccess.registerFilterCollection(1, 1);

    on Exception do openJournalExceptionHandler(exception, gotJournal);
    while gotJournal do
        write auditAccess.getJournalName;
        while auditAccess.getNextRecord(
                            entryType,
                            entryObjectType,
                            offset,
                            entryTimeStamp,
                            entrySerialNumber,
                            entryTransactionID,
                            entryOID,
                            entryClassNumber,
                            entryEdition
                            )do
            if not auditAccess.getClassName(entryClassNumber) = "" then
                write auditAccess.getClassName(entryClassNumber) & " was modified at " & entryTimeStamp.String;
            endif;
        endwhile;
        auditAccess.getNextJournal();
    endwhile;
epilog
    delete auditAccess;
    delete dbAdmin;
end;
```

3. Run the method.

   You will see that only the events relating to the **Customer** class of **BankingModelSchema** are displayed.

4. Modify the method to register a collection rather than a class, as shown in the following code fragment.

```
    auditAccess.getJournal(dbAdmin.getCurrentJournalDirectory, FirstJournal, offset);
    auditAccess.loadDescriptionByName("description<timestamp>.txt");
//  auditAccess.registerFilterClassName("BankingModelSchema", "Customer");
//  auditAccess.registerFilterCollection(1, 1);
    auditAccess.registerFilterClass(1);
    auditAccess.registerFilterCollectionName("BankingModelSchema", "Bank", "allBankAccounts");
```

5. Run the method.

   You will see that only the events relating to the **allBankAccounts** collection of the **Bank** class are now displayed.