JADE Automated Test Code Generator

Version 2018

Contents

JADE Automated Test Code Generator

Introduction

The Automatic Test Code Generator (ATCG) is a JADE schema that enables you to record and replay GUI actions in JADE applications. It does this by capturing the execution of GUI event methods and generating code to replay those actions.

As the test code generated by ATCG is JADE code, you can modify it to meet your requirements.

Loading ATCG

ATCG is available on the **JadeSoftwareNZ** GitHub and you can load it directly from the JADE development environment using the JADE Git integration.

Git works by *cloning*, to create a local copy of a remote repository and then providing tools to manage synchronization and versioning between your local copy and the remote repository. For this module, we use only the Git clone function to obtain a copy of the ATCG system from the **JadeSoftwareNZ** GitHub repository onto your local machine.

When the schema (.**scm**) and Device Data Exchange form and data definition (.**ddx**) files have been cloned, you can load them into your JADE database using a standard schema load.

Note DDX files, implemented in JADE 2018.0.01, serve the same purpose as DDB files except that they are human-readable and are therefore more appropriate for the Git technology, which relies on the ability to compare the differences in files.

Method Recording

ATCG generates its test code by tracking method calls (including **click** events on buttons, and so on) as a user performs GUI actions.

It then generates code fragments (snippets) that call those methods and bundles them into a class (.**cls**) file that is loaded into the database.

When the ATCG test is run, it simply calls the methods that were recorded into its .**cls** file, which has the same effect as performing the GUI actions manually. However, as it does not actually perform any click or mouse actions, the location of elements changing would not affect the test but the naming of the elements changing would.

Exercise 1 – Loading ATCG

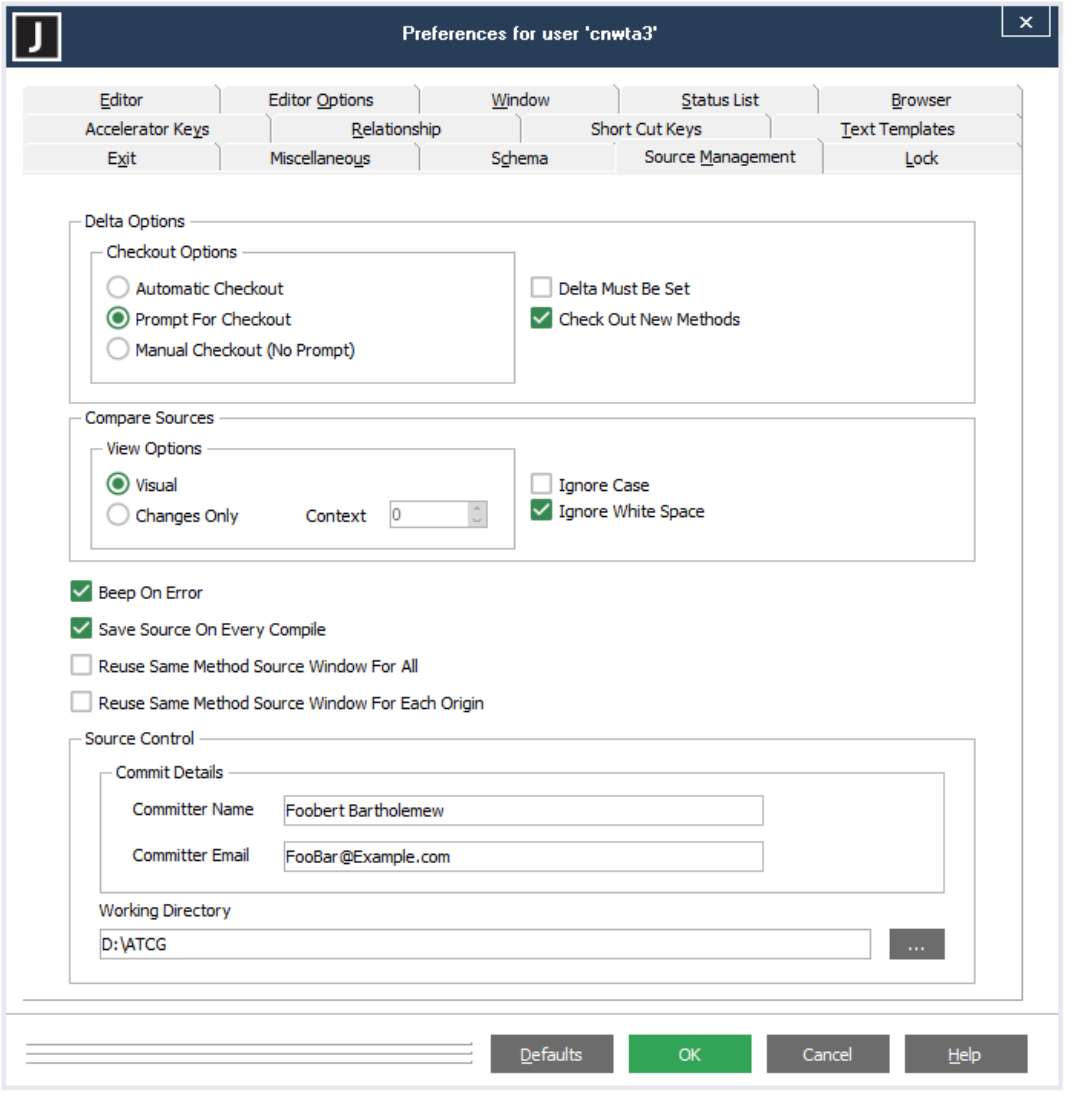
In this exercise, you will use JADE's Git integration to load the ATCG schema into your database.

1. Select the **Preferences** command from the Options menu and in the controls in the Source Control group box on the **Source Management** sheet, add:

   a. Your name

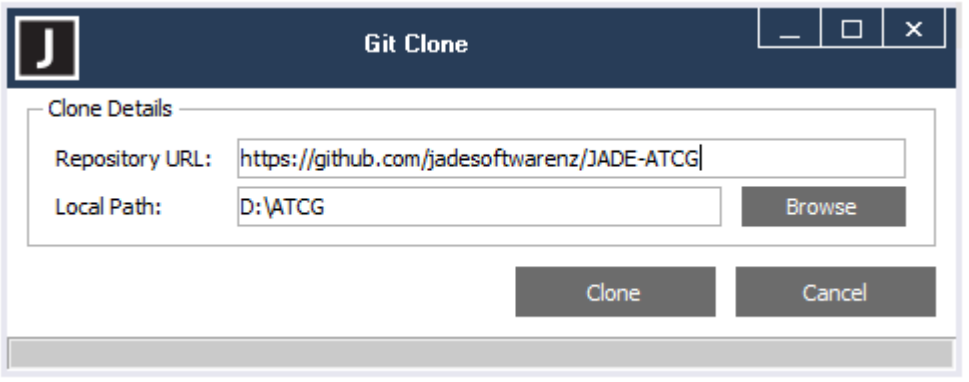
   b. Your email address

   c. A valid directory



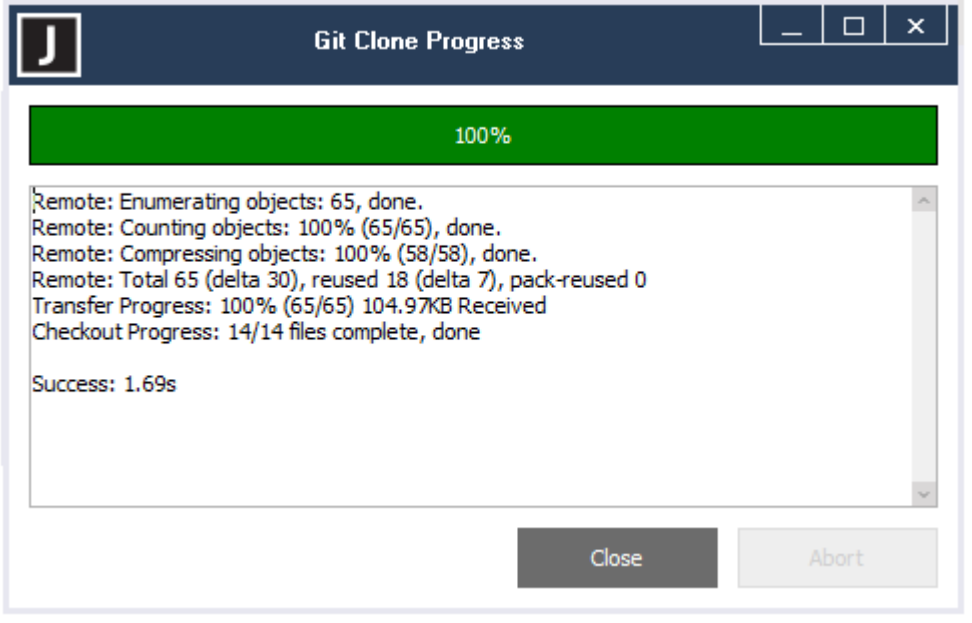
Note The specified working directory will become the folder used for keeping a local copy of the ATCG schemas. This folder should be empty, but it can be called whatever you like, as long as the location is memorable.

2. Select the **Clone** command from the Git Source Control Client submenu of the Browse menu, filling out the details as follows.



Note Replace **D:\ATCG** with the working directory you specified in step 1 of this instruction.

3. The Git Clone Progress dialog displays the clone progress as the files are copied to your working directory, and when complete, enables the **Close** button.

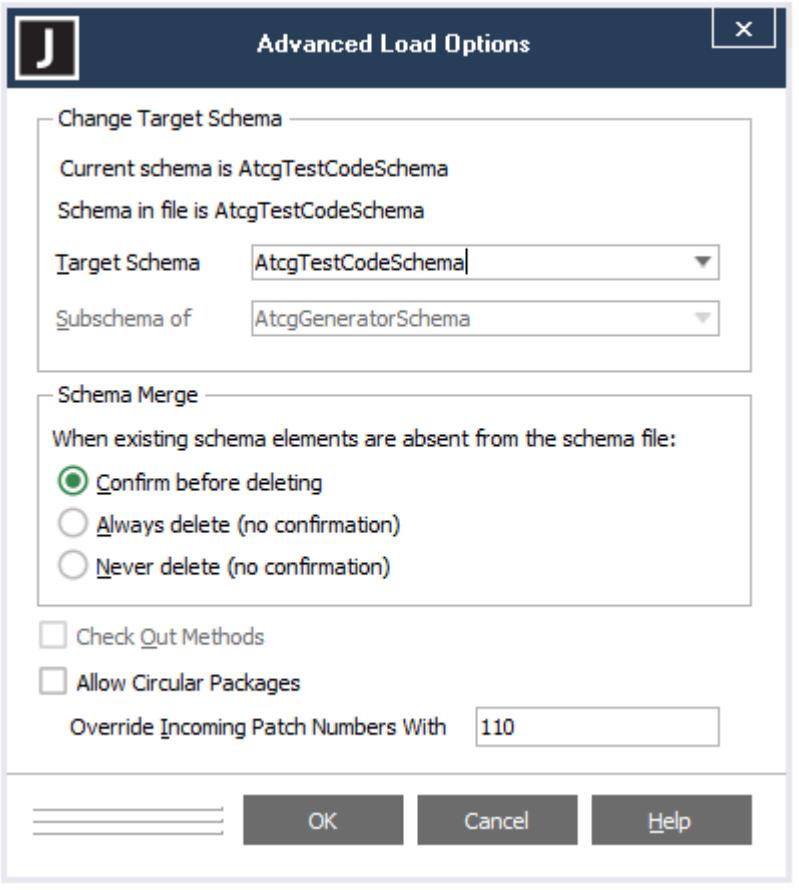


4. Select the **Load** command from the Schema menu. Navigate to your specified working directory, open its **ATCG** subfolder, and then select **AtcgGeneratorSchema.scm**.

5. Click **Advanced** button at the right of the **Load Style** combo box and then fill out the Advanced Load Options dialog, as follows.

   a. Enter **AtcgTestCodeSchema** as the name of the target schema.

b. Select **AtcgGeneratorSchema** as the superschema



6. Click **OK** on the Advanced Options dialog and then click **OK** on the Load Options dialog.

Exercise 2 – Configuring ATCG for the Banking System

In this exercise, you will configure ATCG to track GUI events in the Banking System so that the GUI actions can be replayed.

1. By default, JADE blocks the tracking of user actions. To enable method tracking, add the **MethodTrackingEnabled=true** parameter to the [JadeSecurity] section of your JADE initialization (**jade.ini**) file.

```
[JadeSecurity]
MethodTrackingEnabled=true
```

2. In the **AtcgTestCodeSchema**, find the **XxxProfile** class, which is a subclass of **AtcgProfile**, and rename it to **BankingProfile**. This is the class from which all generated test classes are subclassed.

3. In the **AtcgTestCodeSchemaApp** subclass of **Application**, find and replace all instances of **XxxProfile** with **BankingProfile** (that is, within the **atcgGetControlOptions** and **atcgRecordAppInit** methods).

4. In the **atcgGetControlOptions** method, replace **XxxSchema** with **BankingViewSchema**.

```
// list of schemas to be tracked during recording
targetSchemas.add("AtcgTestCodeSchema");
targetSchemas.add("BankingViewSchema");
```

5. To make testing with ATCG easier, temporarily disable the **getAndValidateSecurity** method in the **GBankingViewSchema** subclass of **Global** in the **BankingViewSchema**, by adding **return true;** as the first line, as follows.

```
getAndValidateUser(usercode: String output; password: String output): Boolean;

vars
    form: Logon;
begin
    return true;
```

6. Navigate to the **BankingProfile** class under **AtcgProfile** in the **AtcgTestCodeSchema**, and then add a new reference property called **form** of type **MainMenu**.

7. Modify the **startup** method of the **BankingProfile** class, as follows.

```
startup() updating;
//  This method will start EVERY profile.
//  It creates the app's mainForm and logs in
vars
    mm : MainMenu;

begin
    app.atcgLogMessageTC("===========");
    app.atcgLogMessageTC("Starting profile");
    app.atcgLogMessageTC("===========");

    // create and show main form, and logon
    create self.form transient;
    self.form.show();

    app.atcgLogMessageTC(method.qualifiedName&" finished");
end;
```

8. Modify the **stop** method of the **BankingProfile** class, as follows.

```
stop() updating;

begin
    app.atcgLogMessageTC("============");
    app.atcgLogMessageTC("Stopping profile");
    app.atcgLogMessageTC("============");

    // log out, close all forms, close main form
    delete self.form;

    app.atcgLogMessageTC("Stopped");
end;
```

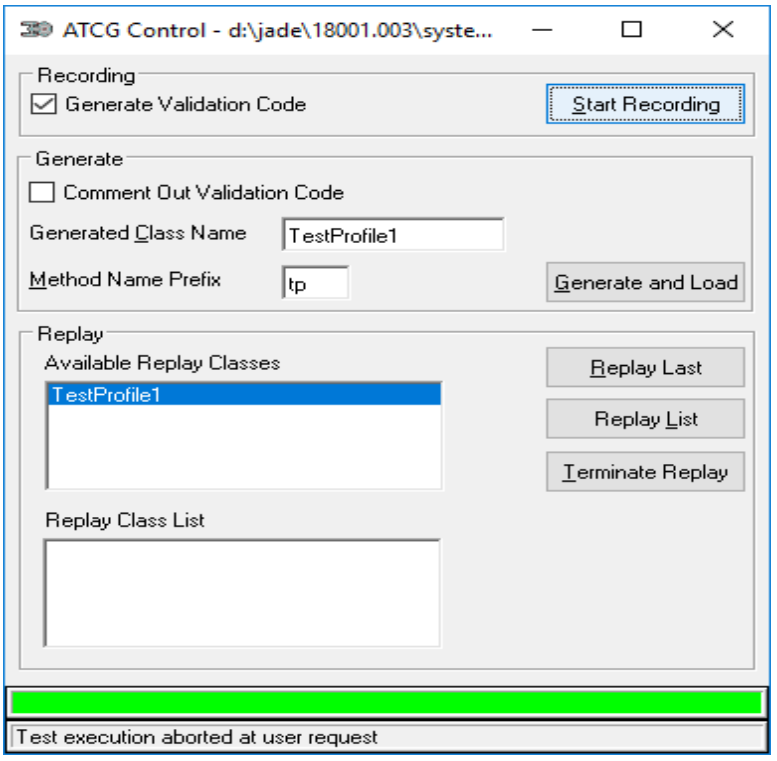
9. Run the **AtcgControlApp** application in **AtcgTestCodeSchema**, to ensure it opens correctly.

Exercise 3 – Recording a Test Run

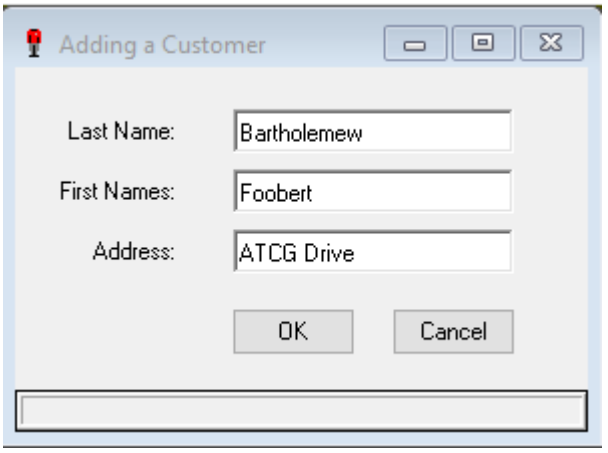
In this exercise, you will use ATCG to record the adding of a customer with the GUI form, creating a test case with the recording, and then replaying it multiple times.

Note Although this is not an efficient way to add many customers, it a good demonstration of ATCG's ability to automate GUI processes.

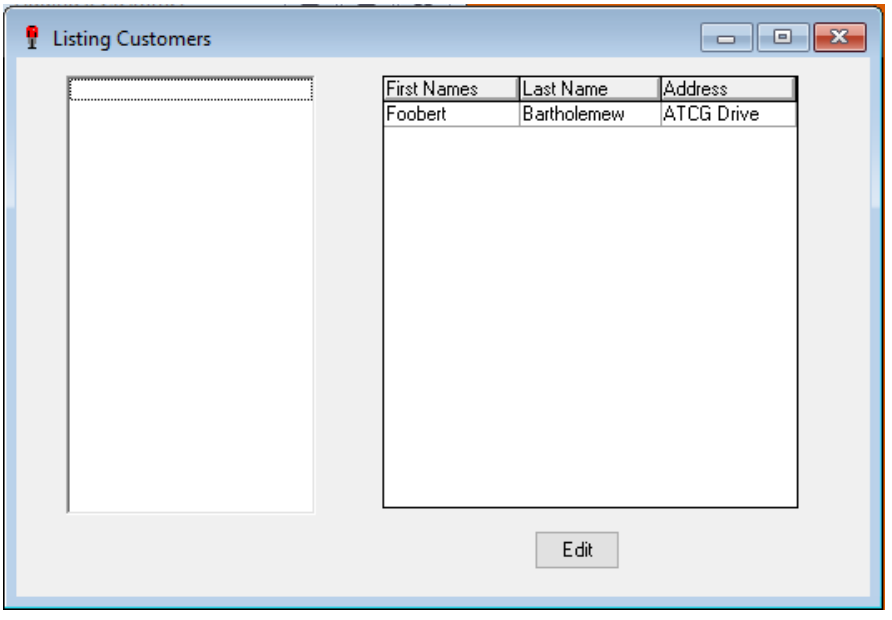
1. Before you record the adding of customers, run the JadeScript **removeTestData** method in the **BankingModelSchema**.
2. Ensure your database is running as a standard (fat) client, as ATCG will not function correctly as a presentation (thin) client or in single user mode.
3. Click **Start Recording** on the ATCG Control dialog.



4. The Banking System then opens. Perform the following actions.
    1. Select the **Add** command from the Customer menu, and then fill out the dialog as follows.

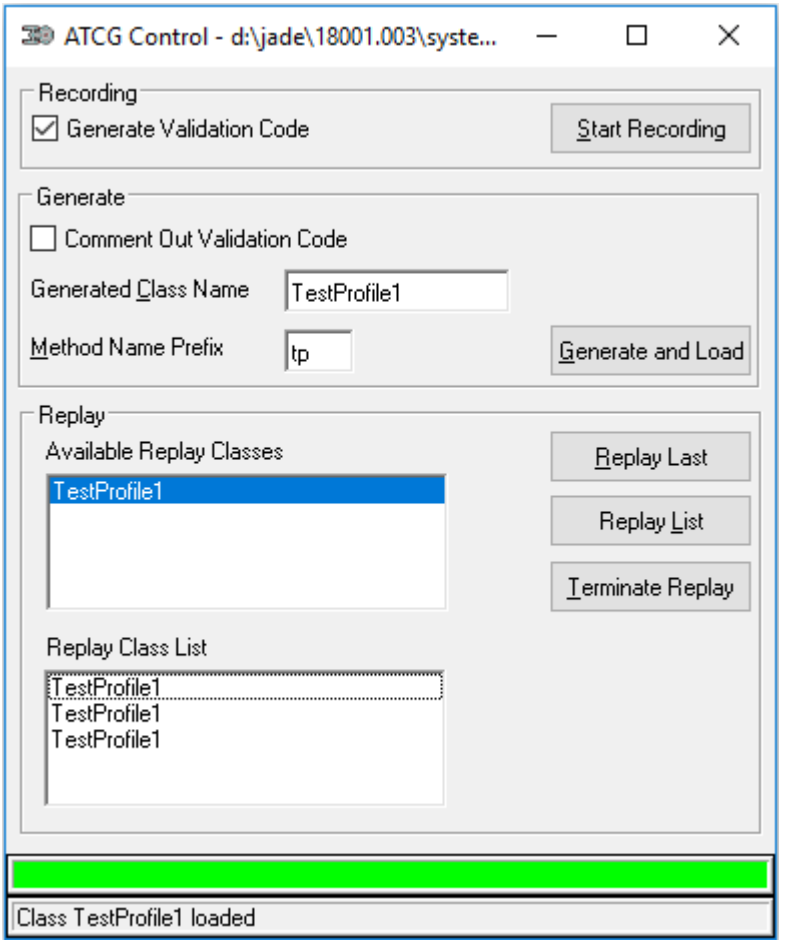


    2. Click **OK**.
    3. Select the **List** command from the Customer menu. You should see one copy of **Foobert Bartholemew** in the list of customers at the right of the dialog.



    4. Close the Banking System form.
5. Click **Generate and Load** on the ATCG Control dialog.
6. When the generation is complete, double-click **TestProfile1** in the **Available Replay Classes** list box to add it to the **Replay Class List** list box.

7. Do this three times, so that there are three copies of **TestProfile1** in the **Replay Class List** list box, as follows.



8. Click **Replay List**.

   ATCG will repeat the same actions you manually performed three times. You should see four copies of **Foobert Bartholemew** in the list of customers: one from the manual run and three from the automated runs.

Generated Code

Whenever a test profile is generated in ATCG, a class with the name of the replay class is added as a subclass of the **AtcgProfile** subclass (which defaults to **XxxProfile** and is renamed **BankingProfile** for the Banking System).

This class has a reference for each form that was used during the test run and a method for each set of actions. In addition, it always has the following three methods.

- **runTest**, which runs each of the methods of the class in sequence
- **startup**, which ensures the preconditions are met
- **tp999_tidyUp**, which ensures the postconditions are met

Although you will not normally want to change these three methods, it can often be useful to manually change the other generated methods to make small changes to the test's operation without the need of a full re-recording.

The manual changing of methods can also be useful to eliminate superfluous actions from the test code. For example, when filling out the Customer Add dialog in the Banking System, you will likely see the generated code look like the following.

```
tp004_CA_btnOK_click():Integer updating;

vars
    keyCode:Integer;
begin
    myCustomerAdd_1:=app.getForm('CustomerAdd').CustomerAdd;
    if unexpected(2, 'myCustomerAdd_1', '', 'FormNotNull', null, myCustomerAdd_1, null, method.qualifiedName) then return 2; endif;
    keyCode:=16;
    myCustomerAdd_1.txtLastName.keyDown(myCustomerAdd_1.txtLastName, keyCode, 1);
    app.doWindowEvents(shortPause);
    myCustomerAdd_1.txtLastName.firstChange(myCustomerAdd_1.txtLastName);
    keyCode:=16;
    myCustomerAdd_1.txtLastName.keyUp(myCustomerAdd_1.txtLastName, keyCode, 0);
    app.doWindowEvents(shortPause);
    myCustomerAdd_1.txtLastName.text:='Bartholemew';
    myCustomerAdd_1.txtLastName.lostFocus(myCustomerAdd_1.txtLastName);
    app.doWindowEvents(shortPause);
    myCustomerAdd_1.txtLastName.change(myCustomerAdd_1.txtLastName);
    keyCode:=16;
    myCustomerAdd_1.txtFirstNames.keyDown(myCustomerAdd_1.txtFirstNames, keyCode, 1);
    app.doWindowEvents(shortPause);
    myCustomerAdd_1.txtFirstNames.firstChange(myCustomerAdd_1.txtFirstNames);
    keyCode:=16;
    myCustomerAdd_1.txtFirstNames.keyUp(myCustomerAdd_1.txtFirstNames, keyCode, 0);
    app.doWindowEvents(shortPause);
    myCustomerAdd_1.txtFirstNames.text:='Foobert';
    myCustomerAdd_1.txtFirstNames.lostFocus(myCustomerAdd_1.txtFirstNames);
    app.doWindowEvents(shortPause);
    myCustomerAdd_1.txtFirstNames.change(myCustomerAdd_1.txtFirstNames);
    keyCode:=16;
    myCustomerAdd_1.txtAddress.keyDown(myCustomerAdd_1.txtAddress, keyCode, 1);
    app.doWindowEvents(shortPause);
    myCustomerAdd_1.txtAddress.firstChange(myCustomerAdd_1.txtAddress);
    keyCode:=16;
    myCustomerAdd_1.txtAddress.keyUp(myCustomerAdd_1.txtAddress, keyCode, 0);
    app.doWindowEvents(shortPause);
    myCustomerAdd_1.btnOK.mouseDown(myCustomerAdd_1.btnOK, 1, 0, 23, 9);
    app.doWindowEvents(shortPause);
    myCustomerAdd_1.txtAddress.text:='ATCG Drive';
    myCustomerAdd_1.txtAddress.lostFocus(myCustomerAdd_1.txtAddress);
    app.doWindowEvents(shortPause);
    myCustomerAdd_1.txtAddress.change(myCustomerAdd_1.txtAddress);
    myCustomerAdd_1.btnOK.mouseUp(myCustomerAdd_1.btnOK, 1, 0, 23, 9);
    app.doWindowEvents(shortPause);
    app.doWindowEvents(shortPause);
    myCustomerAdd_1.btnOK.click(myCustomerAdd_1.btnOK);  // 16:02:47
    return 0;
end;
```

Most of this code is used to reproduce the subtleties of how the form was navigated; for example, tabbing between fields and the handling of the toggling of the Shift key as text was entered. None of these are required for the ATCG to reproduce the actions, so the method could be stripped to the following.

```
tp004_CA_btnOK_click():Integer updating;

vars
    keyCode:Integer;

begin
    myCustomerAdd_1 := app.getForm('CustomerAdd').CustomerAdd;

    myCustomerAdd_1.txtLastName.text    := 'Bartholemew';
    myCustomerAdd_1.txtFirstNames.text  := 'Foobert';
    myCustomerAdd_1.txtAddress.text     := 'ATCG Drive';
    myCustomerAdd_1.btnOK.click(myCustomerAdd_1.btnOK);

    return 0;
end;
```

Note This method also has the **app.doWindowsEvents** calls (pauses) removed so that it replays faster than a human can see.

ATCG and Persistent Data

A special consideration when using ATCG to automate the testing of JADE GUI applications is that any changes to the database while performing an ATCG test can change the behavior of the GUI application and therefore cause future ATCG runs to fail. For example, if using ATCG to automate the testing of the adding of a customer and the customer name has a uniqueness constraint, the second run of the test will fail.

There are several strategies that can be useful to mitigate this risk and ensure the smooth running of many iterations of ATCG testing.

- Start from a fresh database each run; that is, clear all data from the database before or after each run.
- Perform only actions in an ATCG run that do not impact the database.
- Reverse any persistent data changes within the same ATCG test run; for example, if you add a customer, delete it afterwards.

The nature of the database and the goals of the testing determine which of these strategies is most appropriate.

Recording Strategies

As it can often be time-consuming recording ATCG test runs, and the manual running of applications is prone to human error, you should limit the scope of each test run to as small as possible.

This has several benefits, as follows.

- It is easier to re-record if a major mistake is made
- To change the code manually, it is easier to find the specific place in the test code at which a minor mistake was made
- If the behavior of the application being tested changes, it is easier to isolate the change into a single small test, rather than having to re-record all the functionality

The code generated when recording ATCG test runs does not attempt to discriminate between what is relevant and what is irrelevant (for example, resizing or moving forms or unnecessary clicks) to the test you are trying to run.

It is therefore preferable to perform only actions that are strictly necessary and to avoid superfluous actions that generate extra code in the test (making it harder to manage and slower to rerun during replay).

Exercise 4 – Editing Generated Code

In this exercise, you will modify the code generated in the previous exercise, to remove superfluous actions and change the name of the customer to be added.

1. In the **tp004_CA_btnOK_click** method of the **TestProfile1** class (a subclass of the **BankingProfile** class), remove code until the following remains.

```
tp004_CA_btnOK_click():Integer updating;

vars
    keyCode:Integer;
begin
    myCustomerAdd_1:=app.getForm('CustomerAdd').CustomerAdd;
    if unexpected(2, 'myCustomerAdd_1', '', 'FormNotNull', null, myCustomerAdd_1, null, method.qualifiedName) then return 2; endif;

    myCustomerAdd_1.txtLastName.text:='Bartholemew';
    app.doWindowEvents(shortPause);

    myCustomerAdd_1.txtFirstNames.text:='Foobert';
    app.doWindowEvents(shortPause);

    myCustomerAdd_1.txtAddress.text:='ATCG Drive';
    app.doWindowEvents(shortPause);

    myCustomerAdd_1.btnOK.click(myCustomerAdd_1.btnOK);  // 16:02:47
    return 0;
end;
```

Note As you don't need to add any new code yet, just remove the extra lines of code such as those for **keyDown** events.

2. Modify the string values being assigned to the text boxes, as follows.

```
tp004_CA_btnOK_click():Integer updating;

vars
    keyCode:Integer;
begin
    myCustomerAdd_1:=app.getForm('CustomerAdd').CustomerAdd;
    if unexpected(2, 'myCustomerAdd_1', '', 'FormNotNull', null, myCustomerAdd_1, null, method.qualifiedName) then return 2; endif;

    // myCustomerAdd_1.txtLastName.text:='Bartholemew';
    myCustomerAdd_1.txtLastName.text:='Foolish';
    app.doWindowEvents(shortPause);

    // myCustomerAdd_1.txtFirstNames.text:='Foobert';
    myCustomerAdd_1.txtFirstNames.text:='Barry';
    app.doWindowEvents(shortPause);

    // myCustomerAdd_1.txtAddress.text:='ATCG Drive';
    myCustomerAdd_1.txtAddress.text:='Manual Entry Place';
    app.doWindowEvents(shortPause);

    myCustomerAdd_1.btnOK.click(myCustomerAdd_1.btnOK);  // 16:02:47
    return 0;
end;
```

3. Save the method, then re-run the **Replay Class List** from the **ATCG Control** dialog. The automated runs will now create **Barry Foolish** rather than **Foobert Bartholemew**.

Limitations

While ATCG is a powerful tool, there are some limitations that need to be accounted for when using it to create automated GUI tests.

- Exception handling - ATCG does not support the replaying of **Ex_Abort_Action**. Any exception handler that uses this will cause the ATCG recorder to terminate recording at this point.
- Although the **AtcgRecordApp** application must be run as a standard (fat) client, you can run the **AtcgReplayApp** application from a fat client or a thin client.
- You can run one **AtcgRecordApp** application only at a time. However, you can run any number of other applications simultaneously.

- ATCG cannot correctly validate dates or times, as they will vary from recording to replaying. It automatically attempts to detect the presence of a time or date on a form and give a warning, rather than failing with an error. For more details, see "Validation Warnings for Date and Time Fields" in the *JADE Automated Test Code Generator (ATCG) Reference*.
- While calling an event method is usually the same as performing the corresponding GUI action, this can sometimes not be the case. For example, clicking on a table performs the following as well as generating the **click** event.
  - Sets focus to the table
  - Sets the value of the **selected** property
  - Changes the values of the **row** and **column** properties
- The **Form** class **showModal** method blocks execution until the form is unloaded. This prevents ATCG from replaying actions onto that form and it therefore generates a **handleShowModal** method instead (which uses the **show** method rather than the **showModal** method).

Exercise 5 – Troubleshooting Exceptions

This exercise assumes that the state of exception handling in the Banking System is as it was at the end of the JADE Developer's course; that is:

- The **initialize** method arms a global exception handler.

```
initialize() updating;

begin
    on Exception do self.genericExceptionHandler(exception) global;
    self.myBank := Bank.firstInstance();
    if self.myBank = null then
        beginTransaction;
        create myBank persistent;
        commitTransaction;
    endif;
end;
```

- The global exception handler returns **Ex_Abort_Action**.

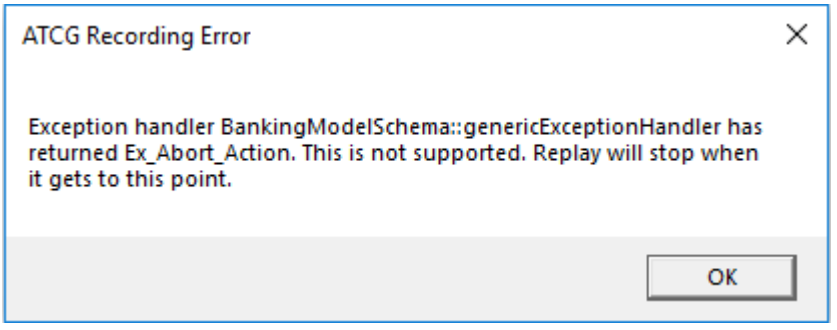
```
genericExceptionHandler(exObj: Exception): Integer;

begin
    abortTransaction;
    exObj.logSelf("errors.log");
    app.msgBox("Unexpected error occurred", "Application Error", MsgBox_OK_Only);
    return Ex_Abort_Action;
end;
```

- The **click** event of the **btnCancel** button on the **Add Customer** form raises an exception.

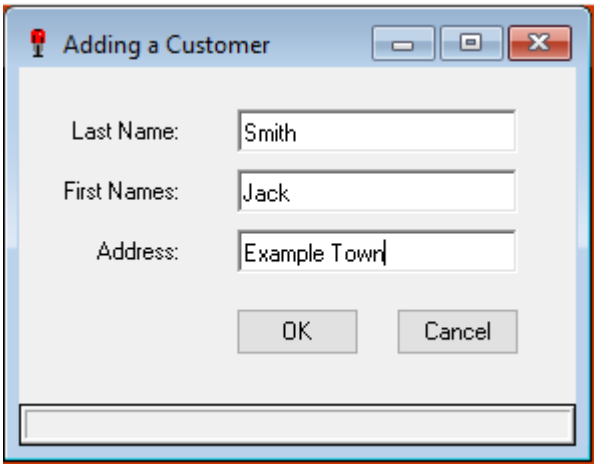
```
btnCancel_click(btn: Button input) updating;

begin
    write 42/0;
    self.unloadForm();
end;
```

In this exercise, you will troubleshoot the handling of exceptions while recording an ATCG test profile. (Note that the end of lines are clipped in some images in this exercise because of long line lengths.)

1. Open the ATCG Control dialog, select **TestProfile2** as the generated class name, and then click **Start Recording**.
2. Open the Customer Add dialog, by selecting the **Add** command from the Customer menu.
3. Click **Cancel**. A message box is displayed, stating that an unexpected error occurred. Click **OK**, and the following should be displayed, because **Ex_Abort_Action** is not supported by ATCG.



4. Close the ATCG Recording Error message box and then add a customer, as follows.



5. Close the Banking System form and then click **Generate and Load** on the ATCG Control dialog.

6. Run the newly created test profile by clicking **Replay Last**. You will see that the test aborts and fails as soon as it gets to the place where the exception was raised.

Tip To fix this, we can either circumnavigate the code paths that generate such an exception while recording or add a local exception handler to the Test Profile method that calls the offending code after generation.

7. Open **AtcgTestCodeSchema** in the Class Browser and navigate to **TestProfile3** under **BankingProfile**.

8. Add a method to this class called **suppressException**, coded as follows.

```
suppressException(e : Exception) : Integer;

begin
    write "Exception suppressed: " & e.text;
    return Ex_Resume_Next;
end;
```

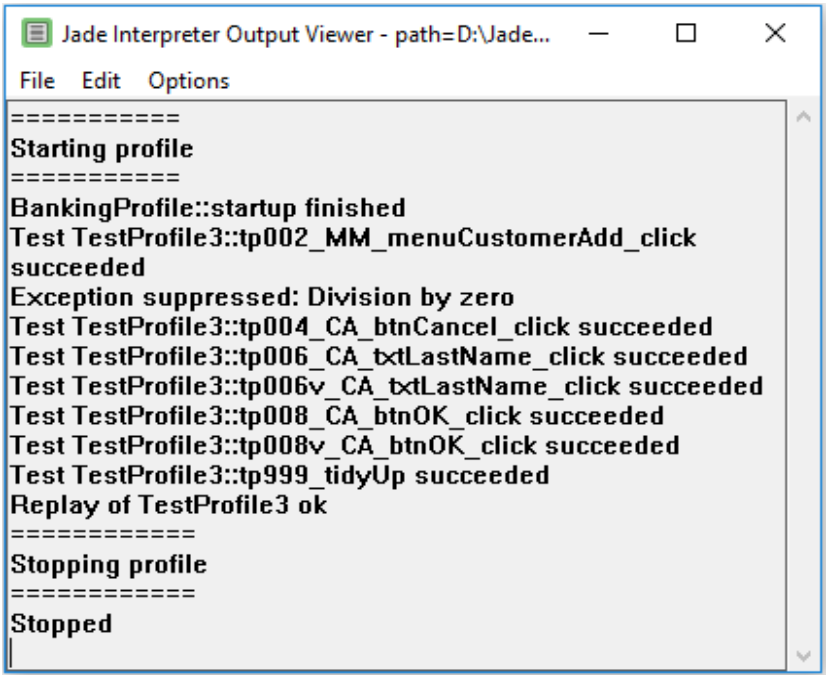
9. Add the following to the **tp004_CA_btnCancel_click** method.

```
tp004_CA_btnCancel_click():Integer updating;

begin
    on Exception do suppressException(exception);

    myCustomerAdd_1:=app.getForm('CustomerAdd').CustomerAdd;
    if unexpected(2, 'myCustomerAdd_1', '', 'FormNotNull', null, myCustomerAdd_1,
    myCustomerAdd_1.btnCancel.mouseDown(myCustomerAdd_1.btnCancel, 1, 0, 24, 7);
    app.doWindowEvents(shortPause);
    myCustomerAdd_1.txtLastName.text:='';
    myCustomerAdd_1.txtLastName.lostFocus(myCustomerAdd_1.txtLastName);
    app.doWindowEvents(shortPause);
    myCustomerAdd_1.txtLastName.change(myCustomerAdd_1.txtLastName);
    myCustomerAdd_1.btnCancel.mouseUp(myCustomerAdd_1.btnCancel, 1, 0, 24, 7);
    app.doWindowEvents(shortPause);
    app.doWindowEvents(shortPause);
    myCustomerAdd_1.btnCancel.click(myCustomerAdd_1.btnCancel);  // 13:01:36
    return 0;
end;
```

10. In the **runTest** method, comment out the following.

```
runTest():Integer updating;

vars
    sa:StringArray;
begin
    create sa transient;
    sa.add('tp002_MM_menuCustomerAdd_click');
    sa.add('tp004_CA_btnCancel_click');
    sa.add('tp006_CA_txtLastName_click');
    sa.add('tp006v_CA_txtLastName_click');
    sa.add('tp008_CA_btnOK_click');
    sa.add('tp008v_CA_btnOK_click');
    sa.add('tp999_tidyUp');
//  app.atcgSetMsgBoxReturn('Unexpected error occurred', 'Application Error', 0, 1);
//  app.atcgSetMsgBoxReturn('Exception handler BankingModelSchema::genericExceptionHandl

    runMethods(sa, 6, "v_");

    app.atcgDetectMissedMsgBoxes;
    return retCode;
epilog
    delete sa;
end;
```

11. Run the **TestProfile3** again. It will now replay the entire test profile successfully.

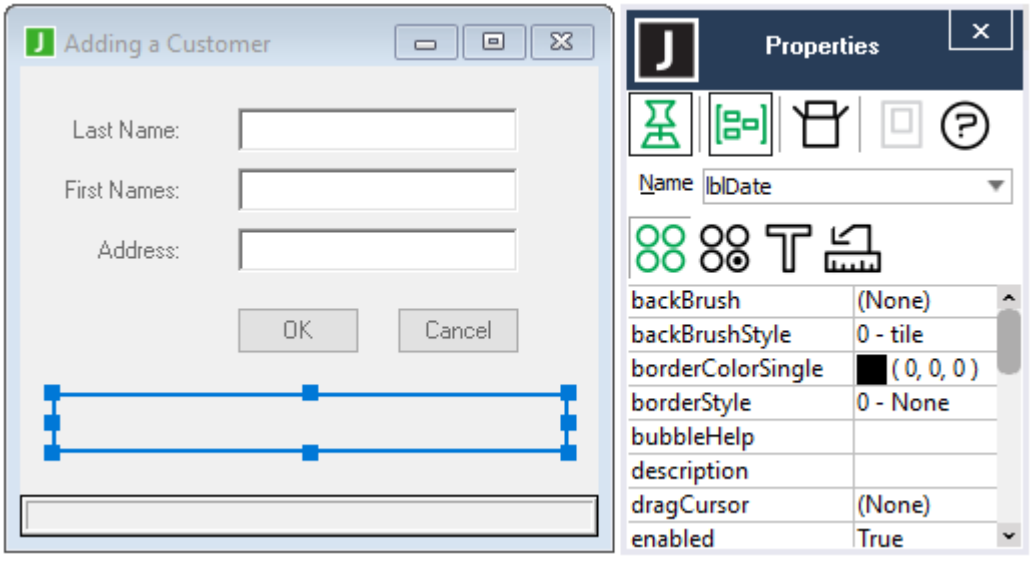


Exercise 6 – Troubleshooting Dates and Times

In this exercise, you will troubleshoot the handling of dates and times in ATCG, which may not remain static over multiple runs of the application.

1. Select **BankingViewSchema** in the Schema Browser and then open the JADE Painter (for example, by using the Ctrl+P shortcut keys).

2. Edit the **CustomerAdd** form by adding a **Label** control called **lblDate** at the bottom of the form.



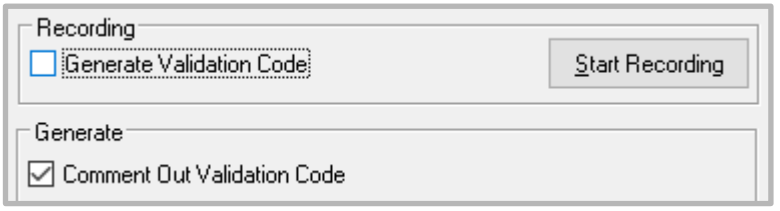
3. Open the **BankingViewSchema** in the Class Browser and modify the **load** method on the **Form Events** sheet of the **CustomerAdd** Form class, as follows.

```
load() updating;

begin
    lblDate.caption := app.actualTime.String;
    txtLastName.maxLength := 0;
end;
```

4. Run the **ATCG Control** from **AtcgTestCodeSchema** and click the **Start Recording** button to begin a new recording.
5. Open the **Add Customer** dialog and click on any field. (As soon as you click on an element in a form, ATCG validates it. Forms that you don't touch are not validated.)
6. Close the Banking System.
7. Change the generate class name to **TestProfile3**.
8. Click **Generate and Load**, to create a new test profile.
9. Click **Replay Last**, to replay the most-recently created test profile. You will notice that the replay fails when it validates and the time part of the **lblDate** doesn't match.

Tip We don't want to validate the time displayed on a form, so will suppress the part of the validation that is checking that element.

10. The simplest, but also least-precise way of suppressing the validation, is to uncheck **Generate Validation Code** to not validate at recording time or to check **Comment Out Validation Code**, to disable all validation at generation time.



You can still go through the code and uncomment the validation, if required.

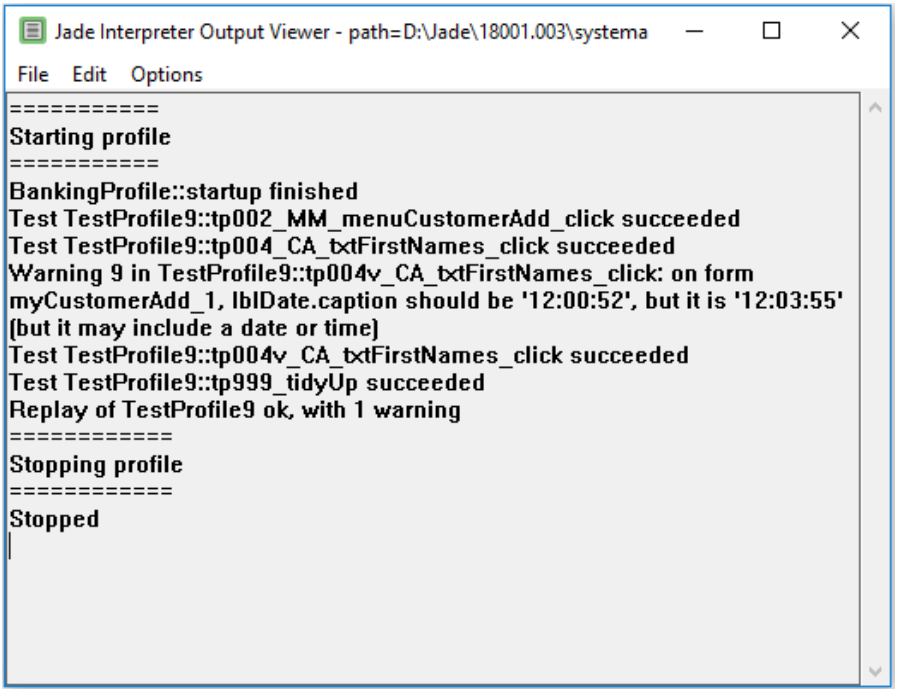
11. The more-precise way of suppressing the validation is to manually find and comment out or delete the offending code.

```
tp004v_CA_txtFirstNames_click():Integer updating;

begin
    if not app.isValidObject(myCustomerAdd_1) then
        return 0;  // presumably the form has been closed by the previous clic
    endif;
    if unexpected(3, 'myCustomerAdd_1', 'txtAddress', 'text', '', myCustomerAd
    if unexpected(4, 'myCustomerAdd_1', 'label3', 'caption', 'Address:', myCus
    if unexpected(5, 'myCustomerAdd_1', 'txtFirstNames', 'text', '', myCustome
    if unexpected(6, 'myCustomerAd_1', 'label2', 'caption', 'First Names:', m
    if unexpected(7, 'myCustomerAdd_1', 'txtLastName', 'text', '', myCustomerA
    if unexpected(8, 'myCustomerAdd_1', 'label1', 'caption', 'Last Name:', myC
//  if unexpected(9, 'myCustomerAdd_1', 'lblDate', 'caption', '11:13:41', myCu
    return 0;
end;
```

12. To enable ATCG to try to auto-detect dates and times, add the following section and parameters to your **jade.ini** file.

```
[ATCG]
ValNoWarningsOverride=false
ValDateTimeWarningOnly=true
```

Running a test profile with a time in it now gives a warning rather than an error and it does not fail the whole test.



Note ATCG's definition for containing a date or time is any string that contains a number followed by a slash or a colon, followed by a second number. This can give both false positives and false negatives, so use with caution.