JADE Logical Certifier

Version 2018

Contents

JADE Logical Certifier

Introduction

The JADE Logical Certifier is a tool for verifying and, if needed, repairing the referential integrity of a JADE database.

Typically, JADE will automatically maintain referential integrity when an inverse relationship is set up within a JADE database and as such, the most common usage of the JADE Logical Certifier is simply to verify that this has been done correctly.

In the unlikely event that somehow an inverse has not been maintained correctly, the JADE Logical Certifier can generate fixes for the various integrity issues an inverse relationship could have.

Referential Integrity

A JADE inverse reference defines a relationship contract between two classes that is enforced on the objects of those classes.

These relationships, which can be one-to-one, one-to-many, or many-to-many, cause JADE to automatically make any references between objects of the contracted classes mutual. For example, consider two classes, **Company** and **Product**, which have the following relationship.

- A **Company** has many **Products**, held in an **allProducts** collection.
- A **Product** has one **Company**, stored in a **myCompany** reference.

As such, we can say that there is a one-to-many relationship between **Company** and **Product**.

With an inverse relationship, we can set a **Company** to automatically add a **Product** to its **allProducts** collection whenever that **Product** sets the **Company** to its **myCompany** reference (or the reverse). With such a relationship set, we can safely say that if a specified **Product** has its **myCompany** set, it is definitely present in the **allProducts** collection of that **Company**.

Broken Referential Integrity

As the use of inverses is both very useful and very reliable, it is common to rely heavily on referential integrity being correct.

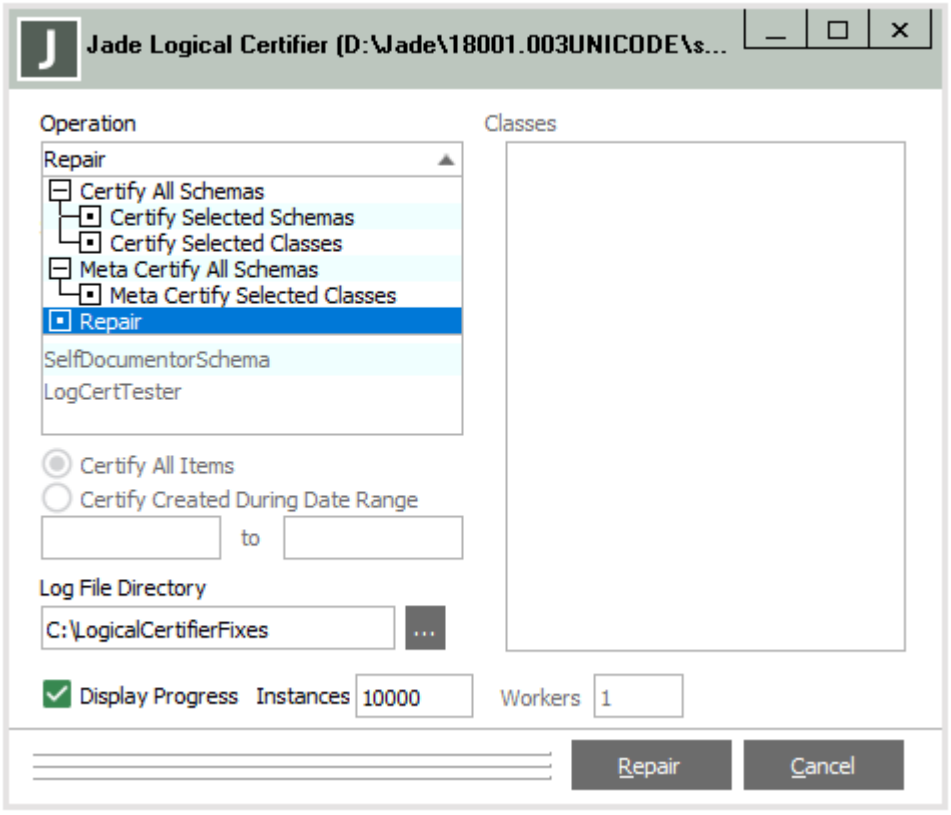
While it is rare for referential integrity to break, any issues have the potential to result in severe consequences; for example, they could result in the accessing of wrong data, the inability to access data at all, or runtime exceptions being generated.

The impact of these events varies from application to application, but it can be significant; for example, if a customer’s details end up stored at the wrong key, another customer could inadvertently gain access to his or her account. As such, even though it is unlikely to see any issue, you should regularly perform certification on user data in production systems.

The JADE Logical Certifier

The JADE Logical Certifier is a RootSchema application that you can open by selecting **RootSchema** in the Schema Browser and then right-clicking the **Run Application** toolbar button.

The Jade Logical Certifier dialog is then displayed.



The JADE Logical Certifier can perform the following functions.

Function	Performs…
Certify All Schemas	A certify on the user data within all schemas of the database and saves the results to the given Log File Directory.
Certify Selected Schemas	A certify on the user data within the selected schema and saves the results to the given Log File Directory.
Certify Selected Classes	A certify on the user data within the selected classes and saves the results to the given Log File Directory.
Meta Certify All Schemas	A certify on the metadata of all schemas of the database and saves the results to the given Log File Directory.
Meta Certify Selected Classes	A certify on the metadata of the selected classes and saves the results to the given Log File Directory.
Repair	The actions prescribed in the **_logcert.fix** file within the given Log File Directory.

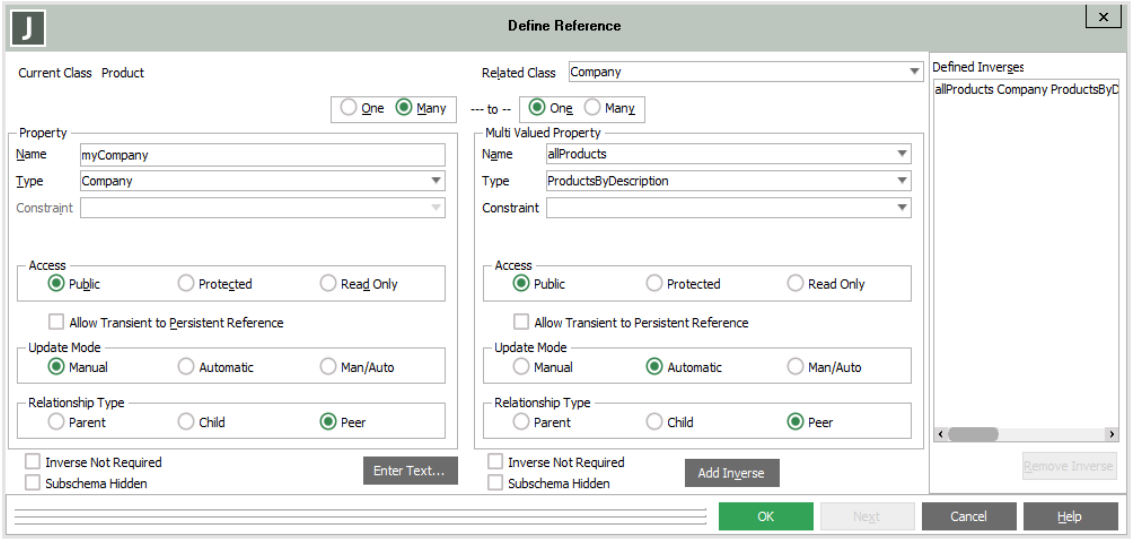
Exercise 1 – Creating a Throwaway Schema

In this exercise, you will create a simple schema with an inverse relationship between two classes so that low-value data intentionally breaks in the next exercise.

1. Create a schema called **LogCertTester**, with two classes called **Company** and **Product**.

> **Caution** While we will only be intentionally breaking *this* schema, if you have important data in another schema, it may pay to take a backup or start from a fresh JADE database, just in case.

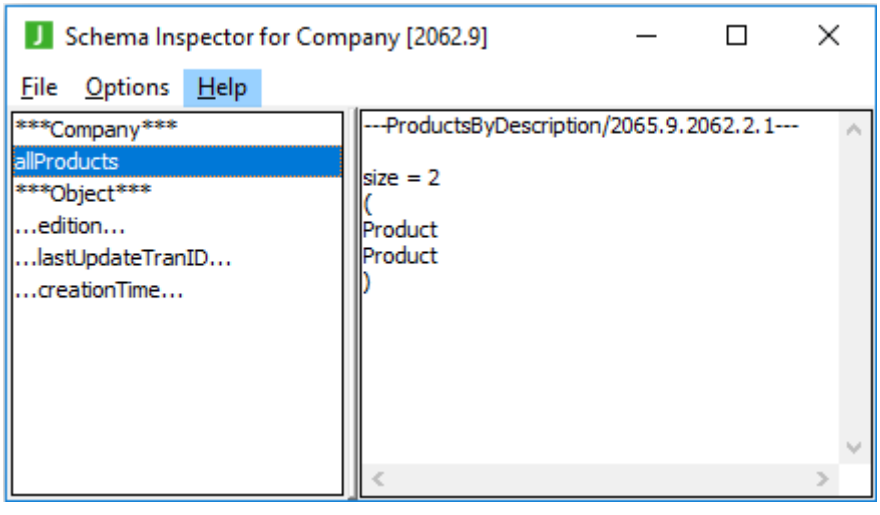
2. In the **Product** class, define a **public** property called **price** of type **Integer**, and a **public** property called **description** of type **String**.
3. Create a subclass of **MemberKeyDictionary** called **ProductsByDescription**, with membership **Product** and the key of **description**.
4. In the **Product** class, define an inverse relationship with **Company**, as follows.



5. Create a JadeScript method called **setup** and code it as follows.

```
setup();

vars
    product : Product;
    company : Company;
begin
    beginTransaction;
    create company persistent;
    commitTransaction;

    beginTransaction;
    create product persistent;
    product.description := "Custom-made Widget";
    product.price := 42;
    product.myCompany := company;
    commitTransaction;

    beginTransaction;
    create product persistent;
    product.description := "Perpetual Motion Machine";
    product.price := 350;
    product.myCompany := company;
    commitTransaction;
end;
```

6. Run the method, checking that the referential integrity has been established by viewing **Company** in the Schema Inspector. The **Company** should have both **Products** automatically added to its **allProducts** collection.



Tip Use the Ctrl+I shortcut keys to open a class in the Schema Inspector.

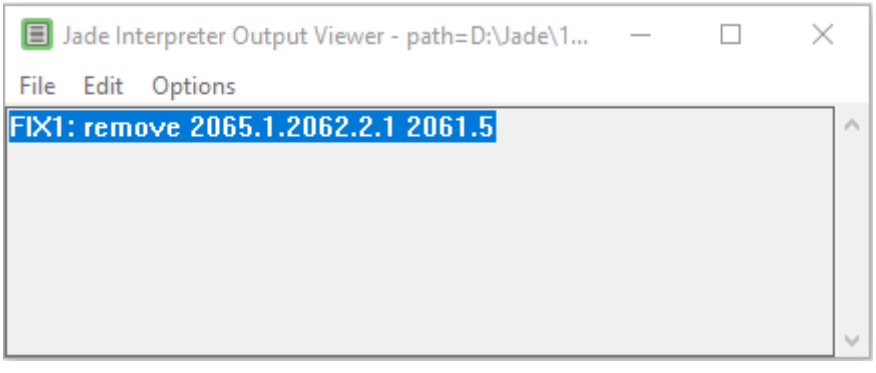
Exercise 2 – Breaking Referential Integrity

In this exercise, you will intentionally break the referential integrity of the **allProducts**/**myCompany** inverse, by removing one of the **Products** from the **allProducts** collection. Normally this would not be possible, as JADE will prevent you from manually modifying an automatic collection involved in an inverse relationship (to ensure referential integrity). However, we can get around this by using the JADE Logical Certifier itself to modify the collection.

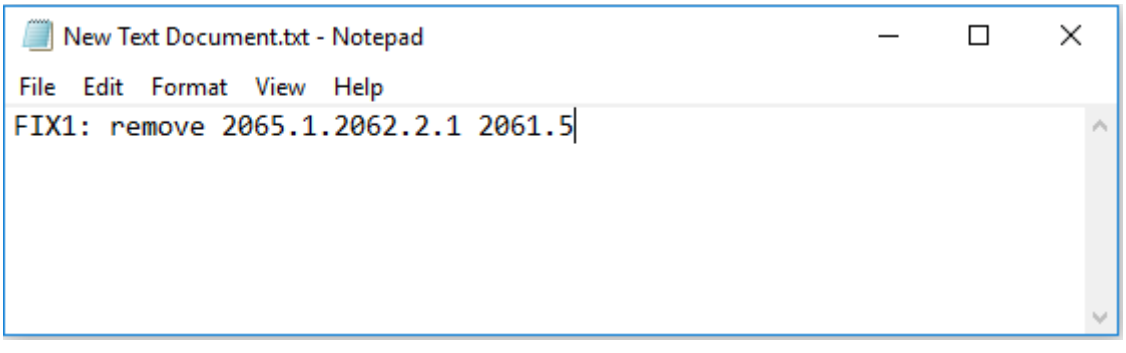
1. Create a JadeScript method called **createBreakingFix** and code it as follows.

```
createBreakingFix();

vars
    fixLine : String;
begin
    //Remove first instance of Product from company's collection of products
    fixLine := "FIX1: remove ";
    fixLine := fixLine & ProductsByDescription.number.String;
    fixLine := fixLine & ".1." & Company.number.String;
    fixLine := fixLine & ".2.1 ";
    fixLine := fixLine & Product.firstInstance.getOidString;
    write fixLine;
end;
```

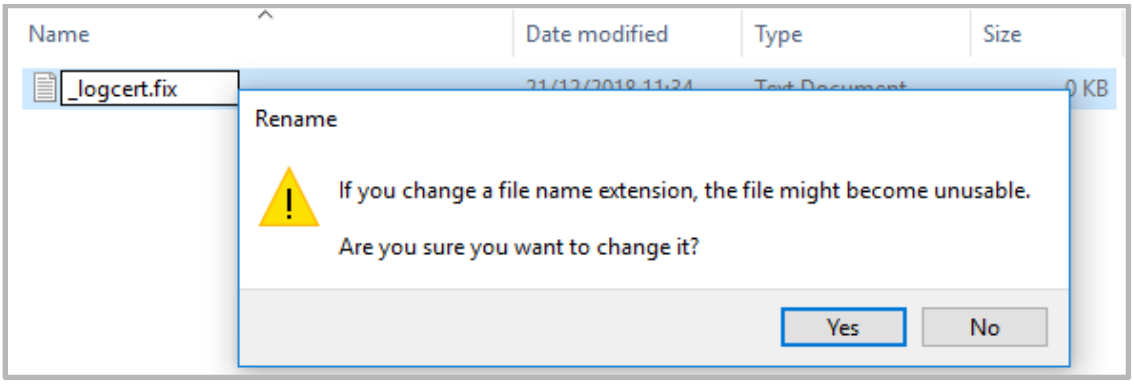
2. Run the method and copy the result from the Jade Interpreter Output Viewer to the clipboard.



3. In your file system, create a folder called **C:\LogicalCertifierFixes**.
4. Create a text document in this folder, paste the result from the **createBreakingFix** method into it, and then save it.



5. Rename the file to **_logcert.fix**.

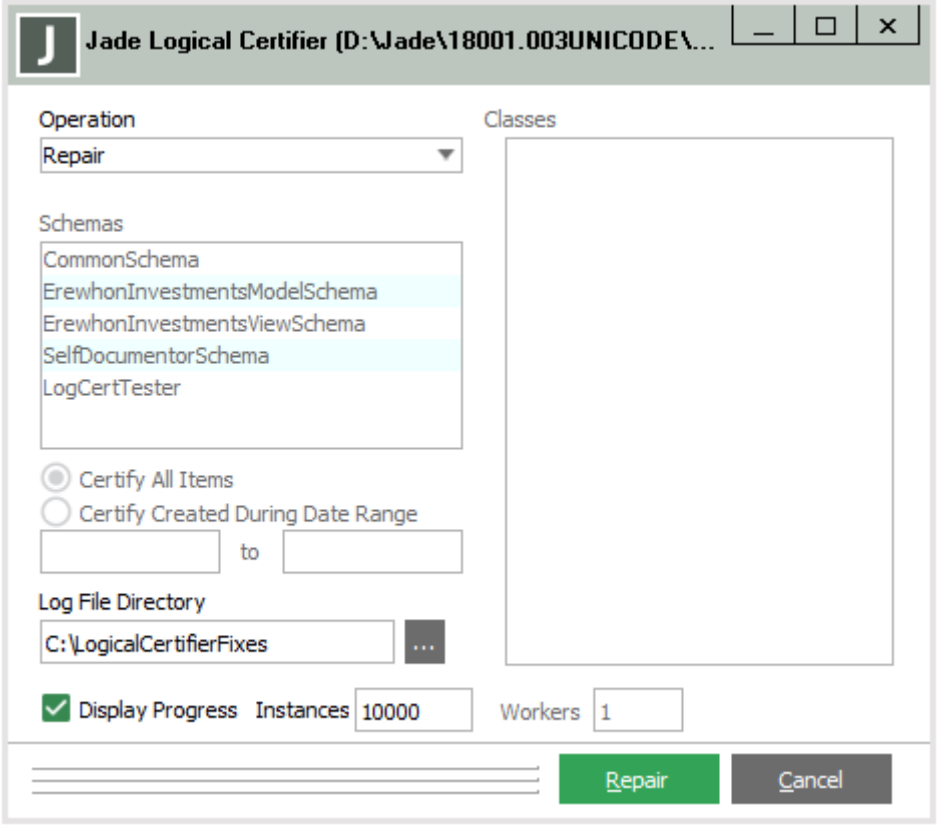


Click **Yes**, to changing the file name extension.

Note You would never normally manually create a **_logcert.fix** file, as they are generated by the certify functions of the JADE Logical Certifier. However, in this exercise, we are actively trying to damage our database.

6. In your JADE development environment, select **RootSchema** in the Schema Browser and open the JADE Logical Certifier by right-clicking the **Run Application** toolbar button.

7. Select **Repair** in the **Operation** combo box, enter **C:\LogicalCertifierFixes** in the **Log File Directory** text box, and then click **Repair**.



8. Inspect the **Company** object in the Schema Inspector. Note that although there is now one entry only in the **allProducts** collection, both **Product** objects still have a reference to **myCompany**.

Fixing Missing Items in Collections

One way that an inverse relationship can be broken is if there is a one-to-many relationship between two classes (for example, a **Company** and its **Products**) where a product holds a reference to its **Company** but that **Company** does not have the **Product** in its collection.

The way that the JADE Logical Certifier fixes this depends on the update mode of the inverse relationship; that is, which class is to be manually updated and which is automatically updated (or if there is a man/auto relationship, where either class can be updated).

The assumption that the JADE Logical Certifier makes is that the manually updated class is correct and the automatically updating class should be the one to change to match. As such, it performs the following fixes for the following cases.

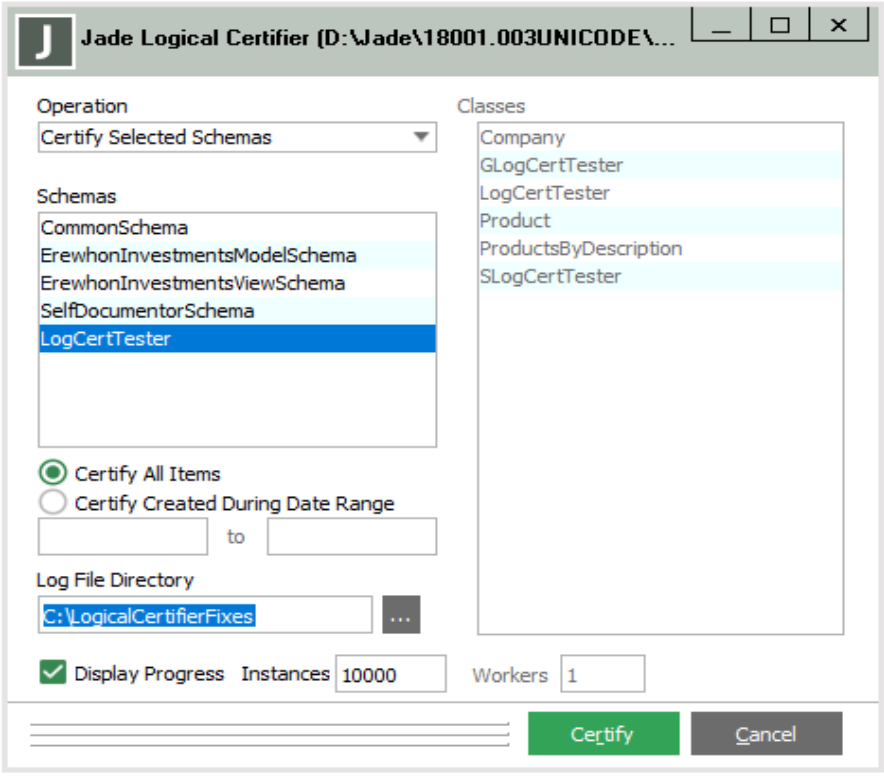
Case	Fix
Manual one to automatic many	Adds the missing object to the collection.
Automatic one to manual many	Keeps the collection as-is and removes the reference to the “one” on the “many” class.
Man/Auto setting	Generates both of the above, but comments them out. You must manually uncomment the fix that is to be done.

Exercise 3 – Repairing Referential Integrity

In this exercise, you will use the JADE Logical Certifier to repair the damage done in the previous exercise.

1. Open the JADE Logical Certifier.

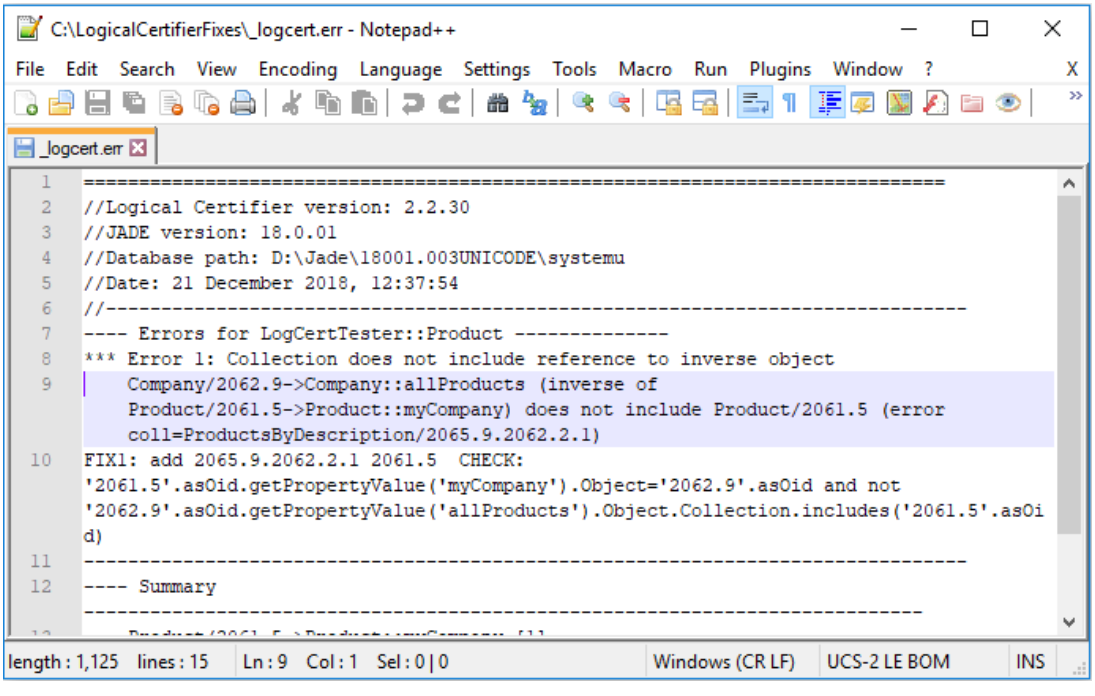
Select **Certify Selected Schemas** in the **Operation** combo box, select **LogCertTester** in the **Schema** list box, and then enter **C:\LogicalCertifierFixes** in the **Log File Directory** text box.



2. Click **Certify** and **Yes** when prompted to confirm that you want to replace the existing file.

   The Certify message box should then be displayed, advising you that the certify operation completed with one error. Click **OK**.

3. To see what error was detected, open the **_logcert.err** file in Notepad++.

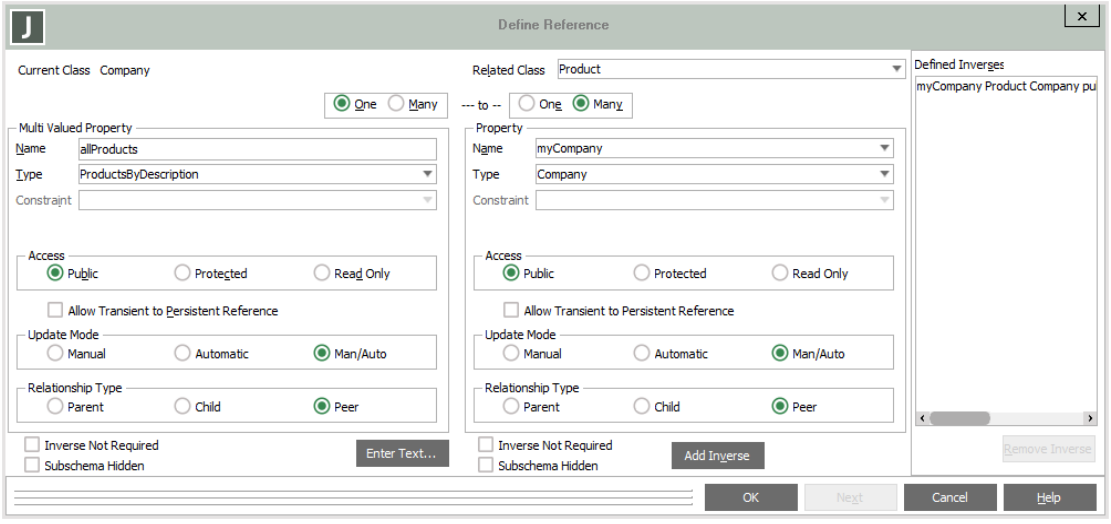


4. From the JADE Logical Certifier, select the **Repair** in the **Operation** combo box and enter **C:\LogicalCertifierFixes** in the **Log File Directory** text box.
5. Click **Yes** on the two subsequent confirmation dialogs.
6. Open **Company** in the Schema Inspector and select the **allProducts** collection. You should see that there are now two **Products** in the collection again.

Exercise 4 - Fixing an Ambiguous Referential Integrity Error

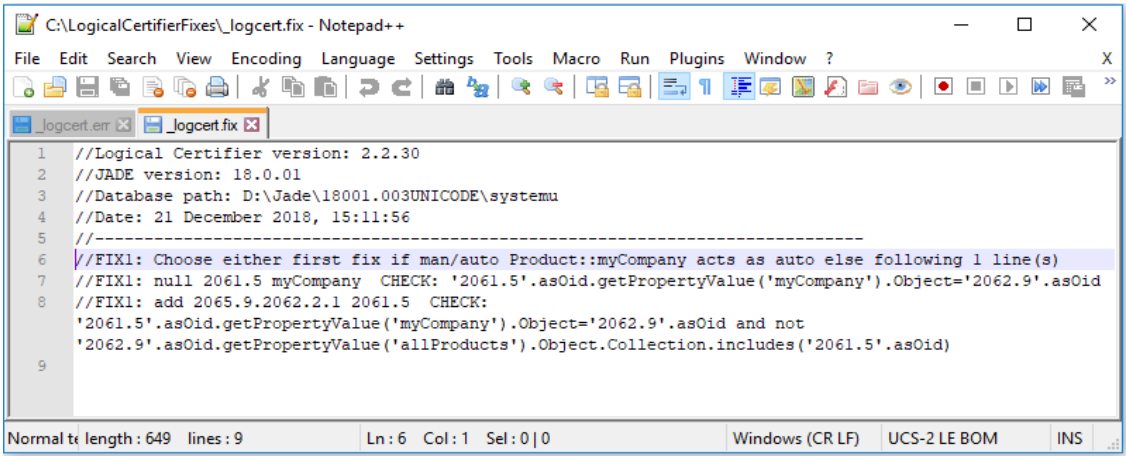
In this exercise, you will set the **Product** / **Customer** relationship to **Man/Auto** and then generate another referential integrity error.

1. Modify the **allProducts** relationship in **Company** to set the update mode to **Man/Auto**, as follows.



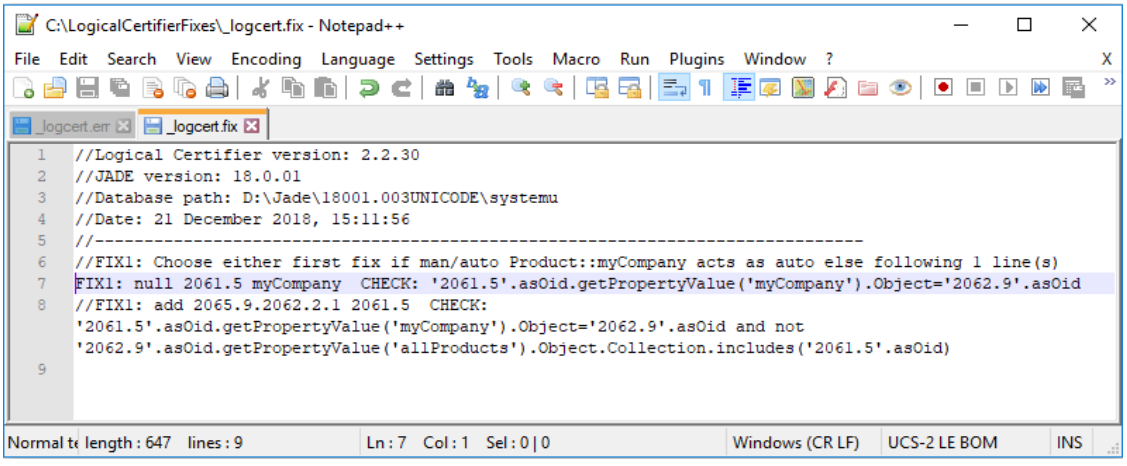
2. Run the JadeScript **createBreakingFix** method and then copy the result to the clipboard.
3. Open the **_logcert.fix** file in the **C:\LogicalCertifierFixes** directory using **Notepad++**, and replace the contents with the result of the JadeScript **createBreakingFix** method that you copied to the clipboard. (Don't forget to save.)
4. Check that the **allProducts** collection in **Company** now contains one product only (using the Schema Inspector).
5. Select **Certify** in the **Operation** combo box and enter **C:\LogicalCertifierFixes** in the **Log File Directory** text box of the Jade Logical Certifier dialog.
6. Open the **_logcert.fix** file in the **C:\LogicalCertifierFixes** directory using Notepad++.

You will see that two fixes have been proposed; that is, setting **myCompany** to null for the **Product** or adding the **Product** to the **allProducts** collection.

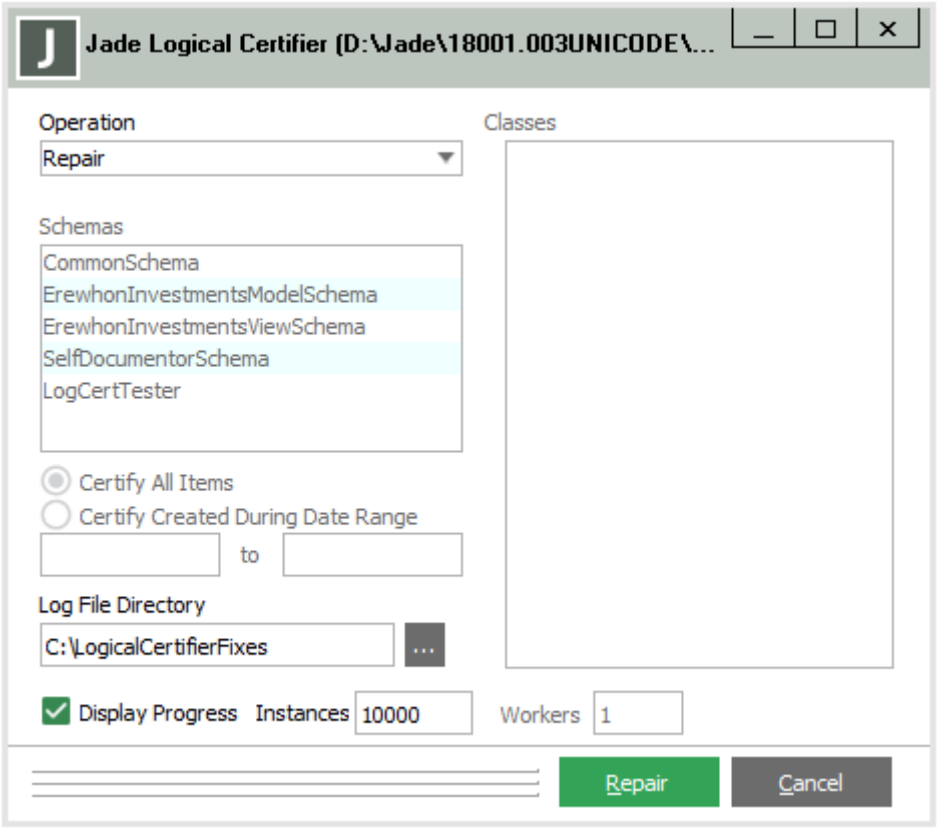


7. To select one of these, simply delete the preceding *//* to uncomment the fix.

   For this exercise, delete the *//* preceding the first proposed fix, as follows.

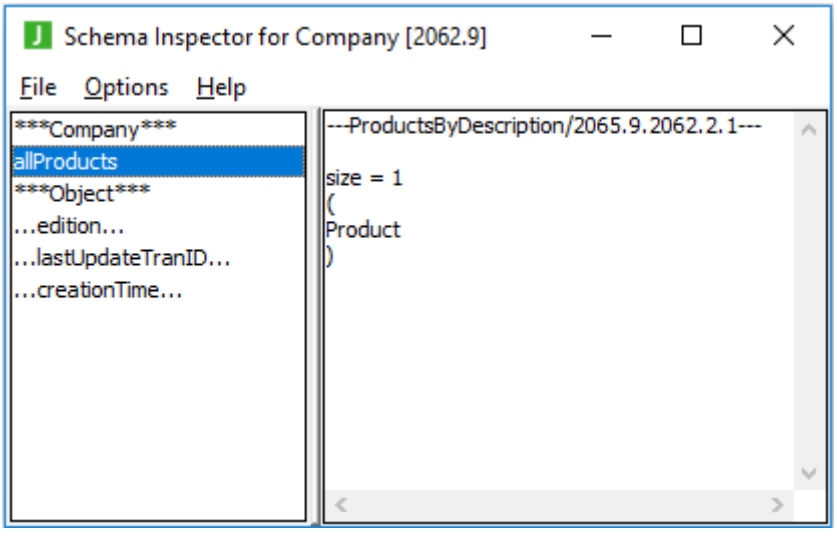


8. Run the **Repair** operation in the Jade Logical Certifier dialog, with **C:\LogicalCertifierFixes** specified in the **Log File Directory** text box.



You will see that it generates no errors, but two warnings. This is simply because the other fix was ignored.

9. Inspect the **allProducts** collection of the **Company**. You will see that it still only has one **Product** in it.



10. Inspect the first **Product** in the Schema Inspector.

    You will see that the **myCompany** reference is a **<null> object reference**; that is, the **Product** has no **Company**.

Handling Invalid Member Keys

One of the most damaging things that can happen to a **MemberKeyDictionary** is when there is a mismatch between the key of an entry and the object referenced by that entry.

This will typically happen only if the database is used while in an erroneous state; for example, consider the following sequence of actions.

1. A **Company** has a **Product** in its **allProducts** collection but that **Product**'s **myCompany** reference is null.
2. That **Product** changes its **description** (that is, its member key), but since the **myCompany** reference is null, the member key is not updated in the **allProducts** collection.
3. The **myCompany** reference on the **Product** is set back to the **Company**. The **Company** now has both the old and the new descriptions as two keys pointing to the same object.

   This could lead to a customer ordering the wrong product or other unexpected behaviors.

Fortunately, the JADE Logical Certifier can fix this, by rebuilding the collection using the **Collection::rebuild** method. This removes the excess entry in the collection and ensures that every member key of the dictionary matches the property of the object referenced in the dictionary.

However, this can fail in the situation where the member key of a **Product** has changed to a key already present in the dictionary. In this case, the step of changing the member key of the dictionary to that of the referenced object's property is impossible, as it would result in a duplicate entry.

Exercise 5 – Creating an Invalid Member Key

In this exercise, you will generate a mismatch between the key of an entry and the object referenced by that entry, and then observe the effect on behavior.

1. Modify the JadeScript **setup** method as follows, and then run it.

```
setup();

vars
    product : Product;
    company : Company;
begin
    beginTransaction;
    Company.instances.purge();
    Product.instances.purge();
    commitTransaction;

    beginTransaction;
    create company persistent;
    commitTransaction;

    beginTransaction;
    create product persistent;
    product.description := "Custom-made Widget";
    product.price := 42;
    product.myCompany := company;
    commitTransaction;

    beginTransaction;
    create product persistent;
    product.description := "Perpetual Motion Machine";
    product.price := 350;
    product.myCompany := company;
    commitTransaction;
end;
```

This restores the test database to its original state.

2. Create a JadeScript method called **breakManualSide** as follows, and then run it.

```
breakManualSide();

vars
    fixLine : String;
begin
    // A product does not know its owning company
    fixLine :=  "FIX1: null " &
                Product.firstInstance.getOidString &
                " myCompany";
    write fixLine;
end;
```

3. Copy and paste the output from the **breakManualSide** method to the **_logcert.fix** file in **C:\LogicalCertifierFixes**.

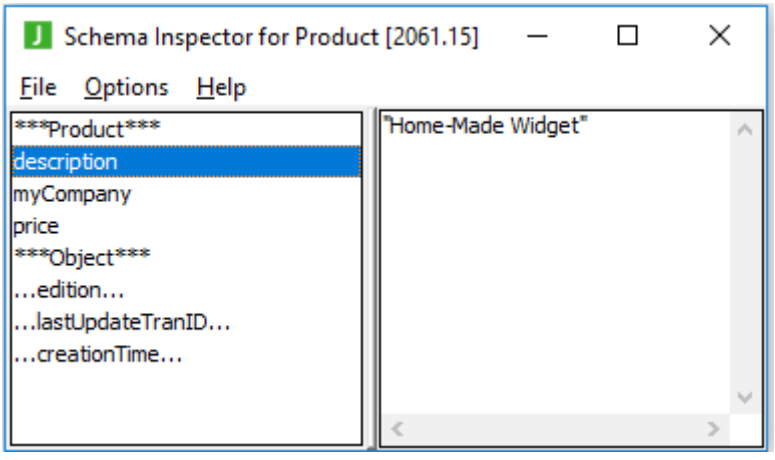
4. Run the **Repair** operation in the Jade Logical Certifier dialog, with **C:\LogicalCertifierFixes** specified in the **Log File Directory** text box.
5. Create a JadeScript method called **changeDescription** as follows, and then run it.

```
changeDescription();

begin
    beginTransaction;
    Product.firstInstance.description := "Home-Made Widget";
    commitTransaction;
end;
```

6. Inspect **Company** in the Schema Collection Inspector (Ctrl+I).
7. Double-click **Company** to view the first (and only) **Company** in the Inspector.
8. Double-click **allProducts** to view the **Company**'s **allProducts** collection.
9. Select the first **Product**. You should see that the description is **Home-Made Widget**.
10. Create a JadeScript method called **findWidget** and code it as follows.

```
findWidget();

begin
Company.firstInstance.allProducts["Custom-made Widget"].inspect();
end;
```

11. Run the **findWidget** method, which brings up the Schema Inspector for a product. If you look at the description, you will see that the **Custom-made Widget** entry in the **allProducts** collection refers to the widget with the description **Home-Made Widget**.



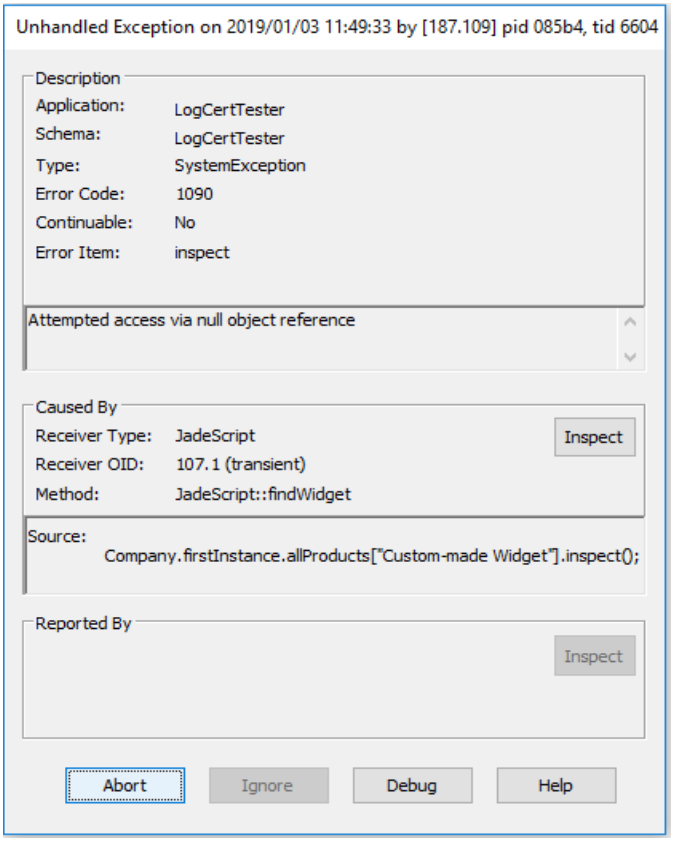
Exercise 6 – Repairing an Invalid Member Key

In this exercise, you will use the JADE Logical Certifier to repair the invalid member key generated in the previous exercise.

1. Run the **Certify** operation in the Jade Logical Certifier dialog to generate a fix in **C:\LogicalCertifierFixes**.
2. Run the **Repair** operation in the Jade Logical Certifier dialog, with **C:\LogicalCertifierFixes** specified in the **Log File Directory** text box.

3. Run the JadeScript **findWidget** method.

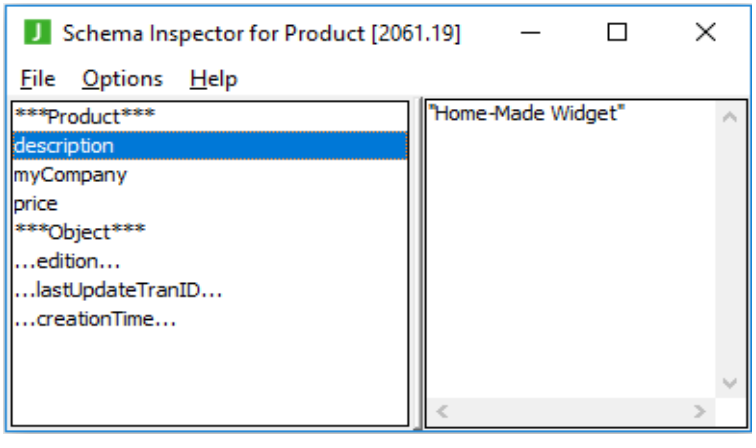
   An unhandled exception 1090 is raised, as **Custom-made Widget** no longer exists as a key in the **allProducts** collection.



4. Modify the **findWidget** method to use the new key, as follows.

```
findWidget();

begin
Company.firstInstance.allProducts["Home-Made Widget"].inspect();
end;
```

5. Run the **findWidget** method, which should now find the **Home-Made Widget** object again.



Failing a Logical Certifier Repair

The JADE Logical Certifier can automatically repair most referential integrity issues that may arise in a JADE database. However, it is possible to get a state where the recommended fix is not possible. An example of this is when there is an invalid member key within a collection, but setting the invalid member key to the correct key property value would result in a duplicate key exception.

The JADE Logical Certifier avoids raising exceptions, so if it is unable to perform a fix, it logs an error in the **_repair.err** file and skips to the next fix in the **_logcert.fix** file, if there is one.

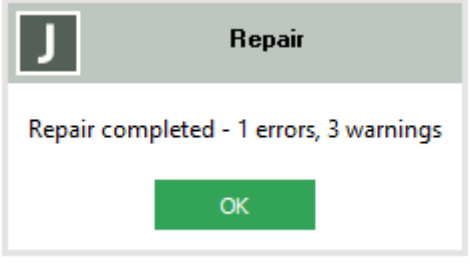
Exercise 7 – Failing a Repair

In this exercise, you will generate a referential integrity error that the JADE Logical Certifier is unable to fix without help.

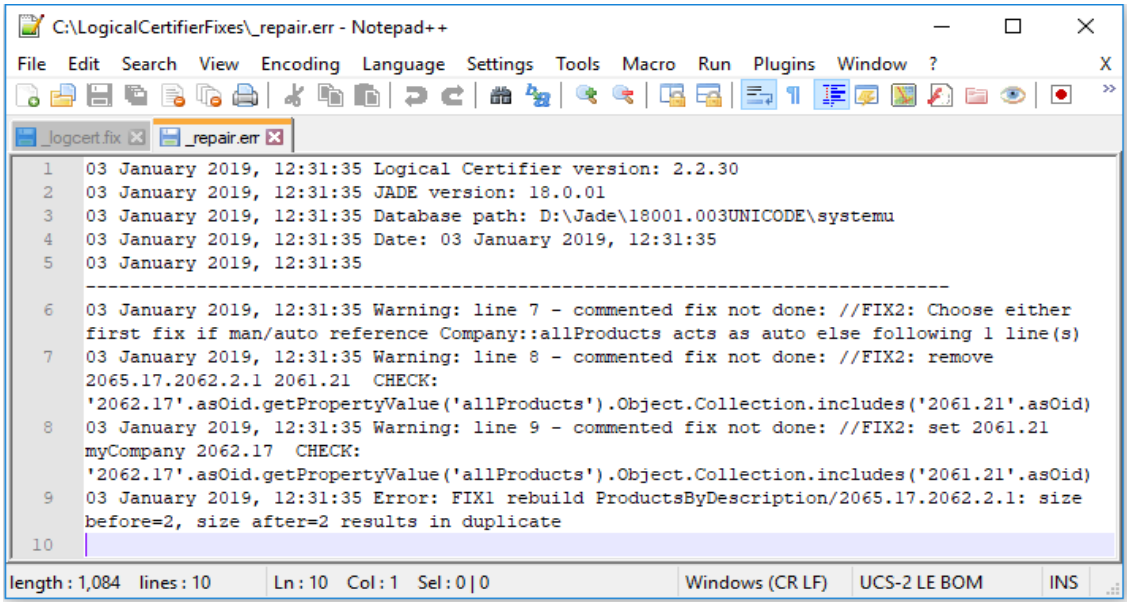
1. Run the JadeScript **setup** method.
2. Run the JadeScript **breakManualSide** method.
3. Copy the output from **breakManualSide** to the **_logcert.fix** file in **C:\LogicalCertifierFixes**.
4. Run the **Repair** operation in the Jade Logical Certifier dialog, with **C:\LogicalCertifierFixes** specified in the **Log File Directory** text box.
5. Modify the JadeScript **changeDescription** method as follows, and then run it.

```
changeDescription();

begin
    beginTransaction;
    Product.firstInstance.description := "Perpetual Motion Machine";
    commitTransaction;
end;
```

6. Inspect **Company** in the Schema Collection Inspector (Ctrl+I).
7. Double-click **Company**, to view the first (and only) **Company** in the Inspector.
8. Double-click **allProducts**, to view the **Company**'s **allProducts** collection.
9. Select the first **Product**. You should see that the description is **Perpetual Motion Machine**.
10. Select the second **Product**. You should see that the description is also **Perpetual Motion Machine**.
11. Run the **Certify** operation in the Jade Logical Certifier dialog to generate a fix in **C:\LogicalCertifierFixes**.
12. Run the **Repair** operation in the Jade Logical Certifier dialog, with **C:\LogicalCertifierFixes** specified in the **Log File Directory** text box. You should see there was one error and three warnings.



13. Open the **_repair.err** file in **C:\LogicalCertifierFixes**.



You can see that the **FIX1** failed, as rebuilding the collection would result in a duplicate. The **FIX2** was skipped, as the **allProducts** collection is set to **man/auto** and therefore there are two possible fixes, both of which are commented out, by default (as in Exercise 4 of this module).

Exercise 8 – Fixing a Failed Repair

In this exercise, you will manually change the data to allow the repair to complete.

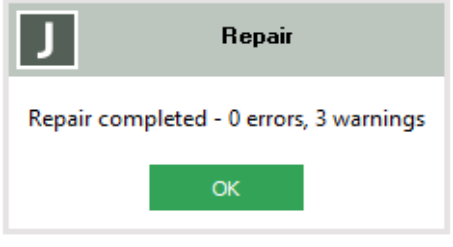
1. Create a JadeScript method called **changeSecondDescription** and code it as follows.

```
changeSecondDescription();

begin
    beginTransaction;
    Product.lastInstance.description := "Temporary Motion Machine";
    commitTransaction;
end;
```

2. Run the **changeSecondDescription** method and then inspect the **allProducts** collection of **Company**.

3. Run the **Certify** operation in the Jade Logical Certifier dialog to generate a fix in **C:\LogicalCertifierFixes**.

4. Run the **Repair** operation in the Jade Logical Certifier dialog, with **C:\LogicalCertifierFixes** specified in the **Log File Directory** text box. You should see there are now zero (0) errors, although there are still three warnings.



5. To verify that the **allProducts** collection now has correct keys, modify the **findWidget** method as follows, and then run it.

```
findWidget();

begin
Company.firstInstance.allProducts["Perpetual Motion Machine"].inspect();
Company.firstInstance.allProducts["Temporary Motion Machine"].inspect();
end;
```

You should see both objects open in the Schema Inspector.