GXtest 演習

GeneXus によるテスト

目次

目的

このトレーニングは、GXtest の主な機能を実践するためのガイダンスです。与えられたプロシージャーに対してユニットテストを作成し、データベースの状態を検証し、テストを実行する際にモックデータベースを使用することを学びます。

本トレーニングは以下の内容で構成されています。

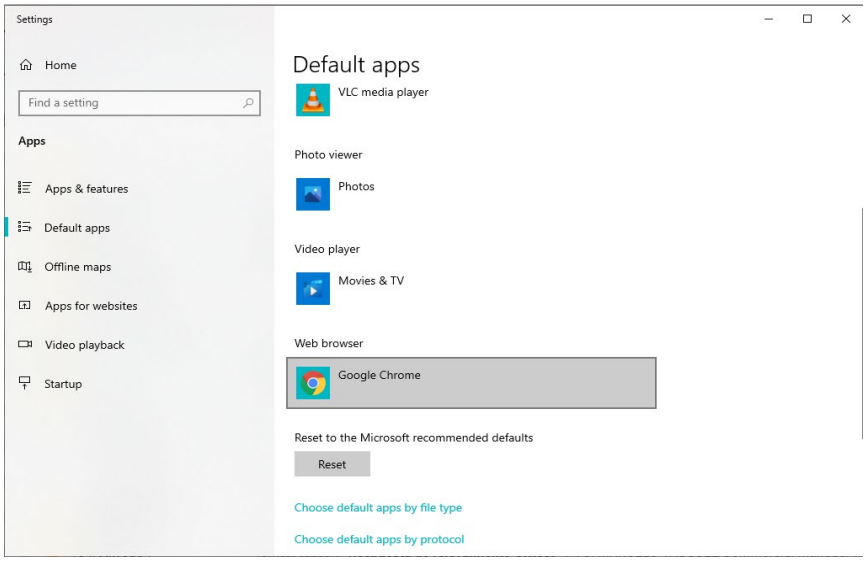
- テストカバレッジ機能の紹介
- REST サービスとして描写されたプロシージャーに対する Rest テストの作成と実行
- GXtest Recorder を使用した UI テストの作成と Web UI テストオブジェクトへのインポート
- Test Suite タイプのオブジェクトの構築

環境セットアップ

トレーニングを始める前に、以下のように環境を整える必要が御座います。

1)Windows のデフォルトの Web ブラウザとして Google Chrome を設定します。

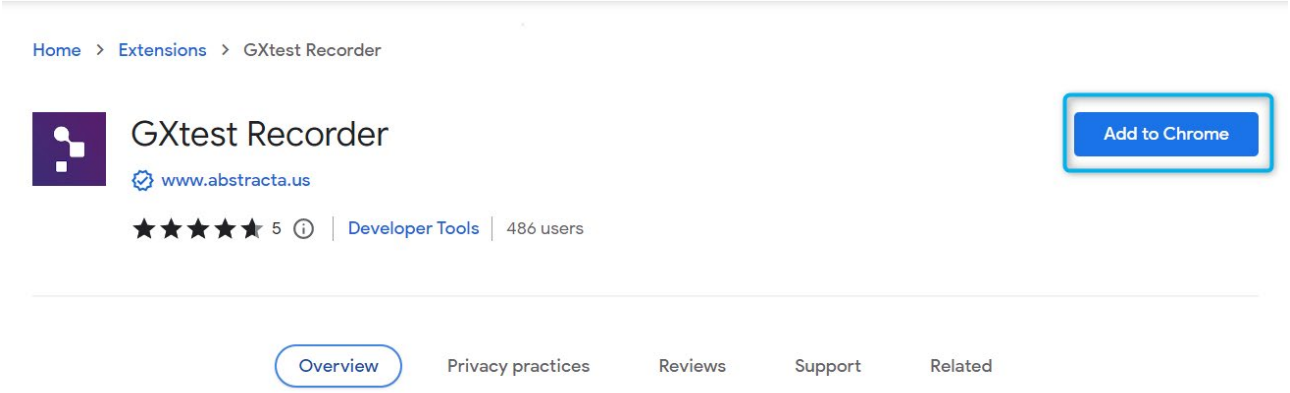
a) “既定のアプリ”の Web ブラウザを”Google Chrome”に設定します。



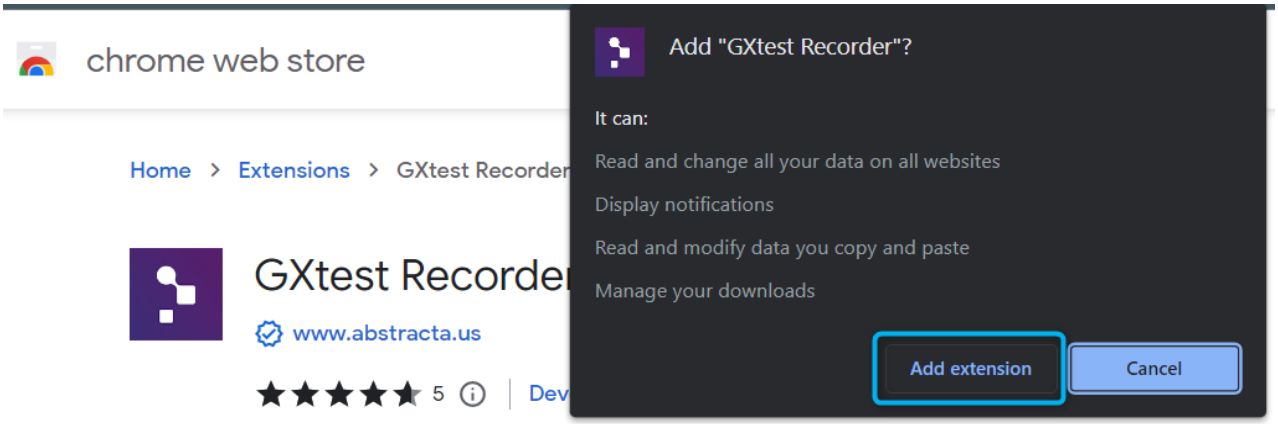
2)GXtest Recorder 拡張機能をインストールします。

a)Chrome ウェブストア内で”GXtest Recorder”を検索

b)「Chrome に追加」を選択



c)”拡張機能に追加”を選択

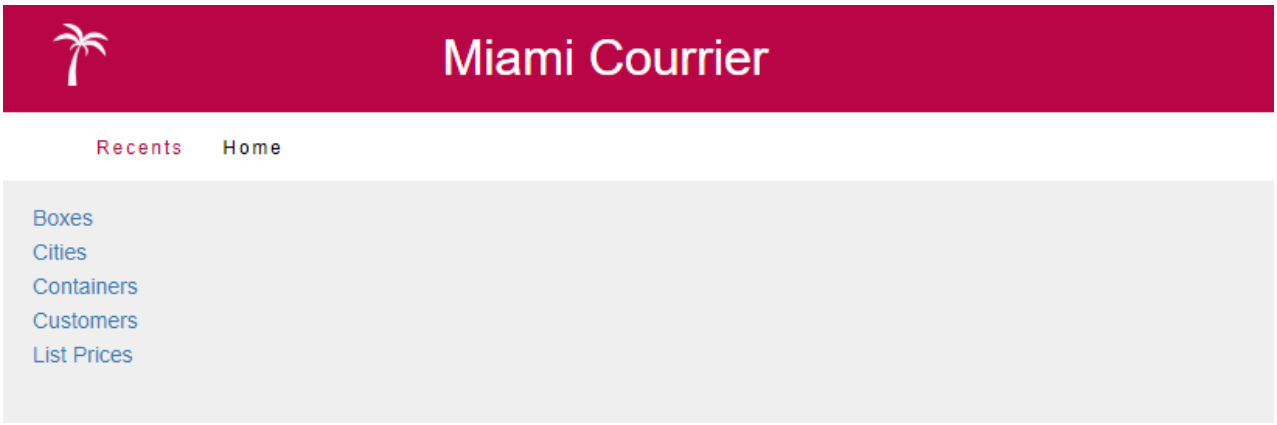


d)Google Chrome を閉じる

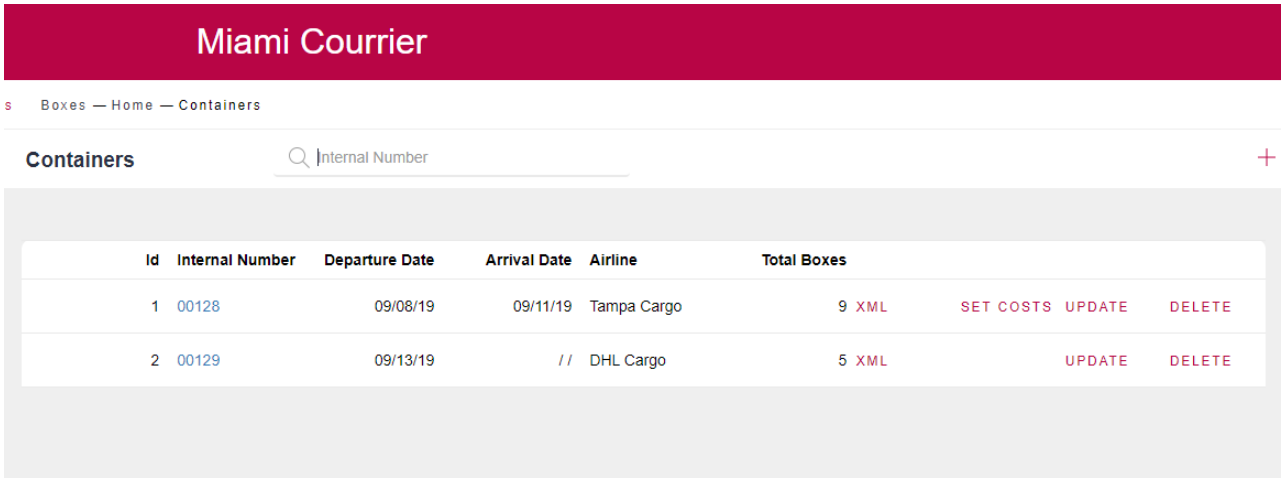
アプリケーション

このアプリケーションは、国際郵便をモデルにしており、輸送コンテナとそれに対応するボックス、顧客、および配送業者を管理します。

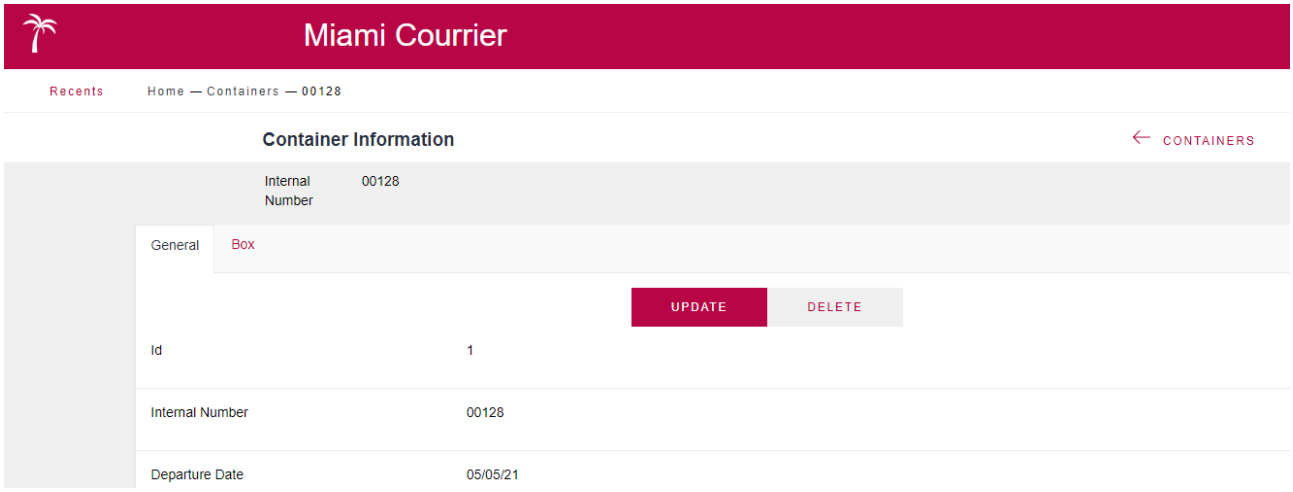
- GeneXus IDE を起動します。
- “GXtestHandsOn"という名前の KB をサーバーからチェックアウトします。
  https://training03.genexusserver.com/courses
  GeneXus Server から「ファイル」→「新規」→「GeneXusServer からのナレッジベース」と進み、「サーバーKB URL」に以下の URL を入力する
  “*https://training03.genexusserver.com/courses*”.
  注：上記サーバーへの接続には GeneXus Account が必要です。GeneXus Account の取得がお済み出ないお客様は同梱の資料「GeneXus Account の取得方法_v1.0.docx」をご確認の上、GeneXus Account を取得して下さい。
- GeneXus フォルダ内の”Home”パネルをスタートオブジェクトとして設定します。
  GeneXus IDE の KB エクスプローラから「Root Module」→「GeneXus」→「Web」に移動し、”Home”パネルを右クリック後「開始オブジェクトとして設定」をクリック
- GeneXus IDE からアプリケーションを実行します（F5）。
  Chrome ブラウザで以下のウィンドウが開きます。:



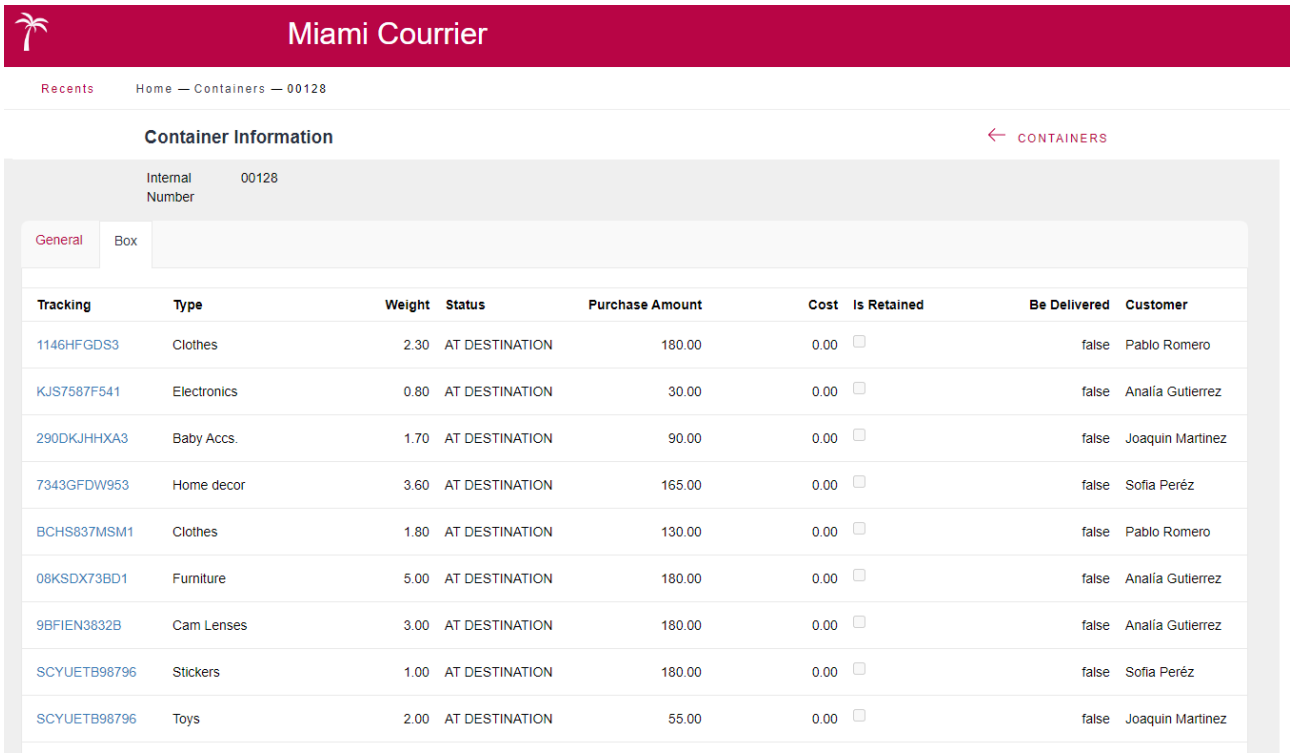
- “Containers"をクリックします。このアプリケーションには、2 つのコンテナとそれに対応するボックスがあります。
  コンテナ 1 には 9 個のボックスが入っていることに着目してください。



- コンテナ 1 のリンクである「00128」をクリックすると、コンテナ情報が表示されます。:



- 「Box」タブをクリックします。すべてのボックスの”Cost”欄が 0 となっています。



Tracking	Type	Weight	Status	Purchase Amount	Cost	Is Retained	Be Delivered	Customer
1146HFGDS3	Clothes	2.30	AT DESTINATION	180.00	0.00		false	Pablo Romero
KJS7587F541	Electronics	0.80	AT DESTINATION	30.00	0.00		false	Analía Gutierrez
290DKJHHXA3	Baby Accs.	1.70	AT DESTINATION	90.00	0.00		false	Joaquin Martinez
7343GFDW953	Home decor	3.60	AT DESTINATION	165.00	0.00		false	Sofia Peréz
BCHS837MSM1	Clothes	1.80	AT DESTINATION	130.00	0.00		false	Pablo Romero
08KSDX73BD1	Furniture	5.00	AT DESTINATION	180.00	0.00		false	Analía Gutierrez
9BFIEN3832B	Cam Lenses	3.00	AT DESTINATION	180.00	0.00		false	Analía Gutierrez
SCYUETB98796	Stickers	1.00	AT DESTINATION	180.00	0.00		false	Sofia Peréz
SCYUETB98796	Toys	2.00	AT DESTINATION	55.00	0.00		false	Joaquin Martinez

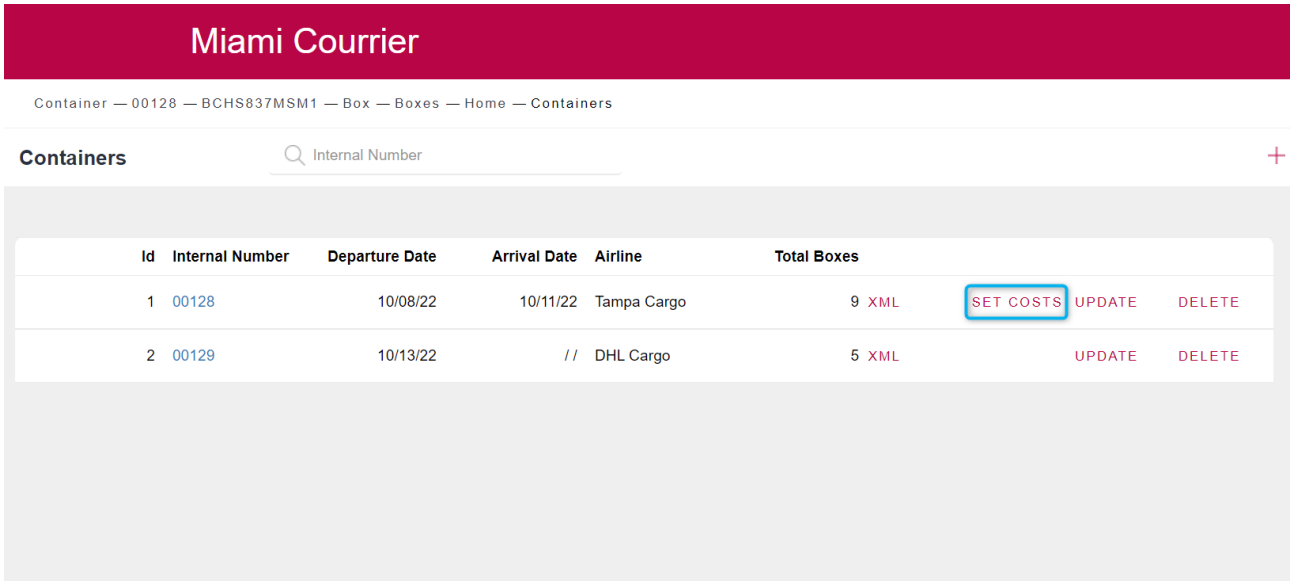
テスト概要

次に、「コンテナ内の箱にコストを設定する」という機能が正しく動作していることを確認します。

この機能は、"Containers"画面のインターフェースレベルで実装されています。

ページ内のリンクである「SET COSTS」をクリックすると、そのコンテナに関連付けられたボックスのコストが設定されます。

なお、この時点で「SET COSTS」をクリックすると、アプリケーションデータベースのステータスが変更されるため、テストの実行前に「SET COSTS」をクリックしないようご注意下さい。



今回は 5 種類の Unit Test シナリオを実行します。:

1)ユニットテスト成功：正しい値を用いて、テストが正常に実行されることを確認

2)ユニットテスト失敗：不正な値を用いて、テストが失敗した際に何が起こるかを確認

3)データベース検証：Unit Test にデータベースのステータス検証を追加する

4)モックデータによる実行：シナリオ 1 で作成した Unit Test に対し、モックデータを記録して実行する。

5)動的データ用のモックファイルの編集：”Set Container Arrival Date”プロシージャーの Unit Test で記録したモックファイルの内容を編集し、Date 型データで何度も実行可能とする

また、シナリオ 1 と同じ手順で、GAM 認証を使用した REST サービスのテストを作成し、実行することを学びます。

そして、その機能をインターフェースレベルで検証する、UI テストを実施します。

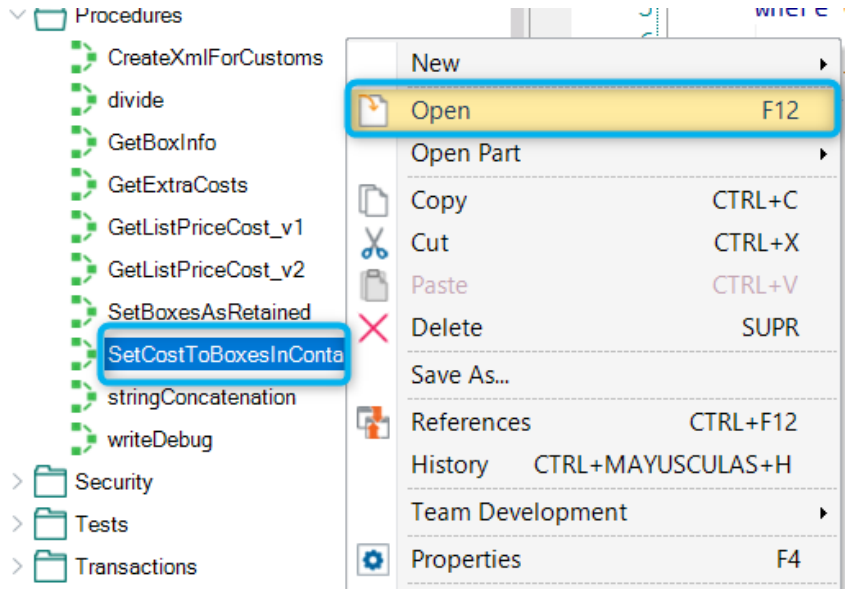
また、コンテナボックスの機能である”Get Extra Costs”に対して作成されたユニットテストのテストカバレッジ分析も含まれており、テストによって実行されるオブジェクトのコードを 100% カバーすることを目標としています。

最後に、以前に作成したテストをグループ化し、特定の順序でユニットとして実行するための Test Suite オブジェクトを作成することを学びます。

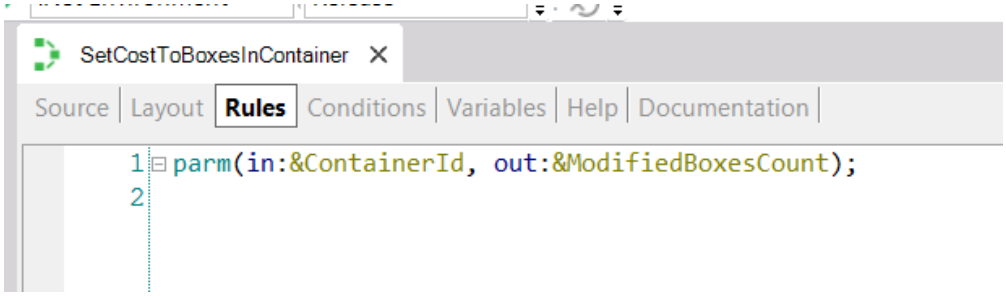
ユニットテスト

このセクションでは、さまざまなユニットテストのシナリオを実行する方法をご案内します。コンテナ内の箱にコストを設定する機能は、KB 内の"SetCostToBoxesInContainer"プロシージャーに実装されています。

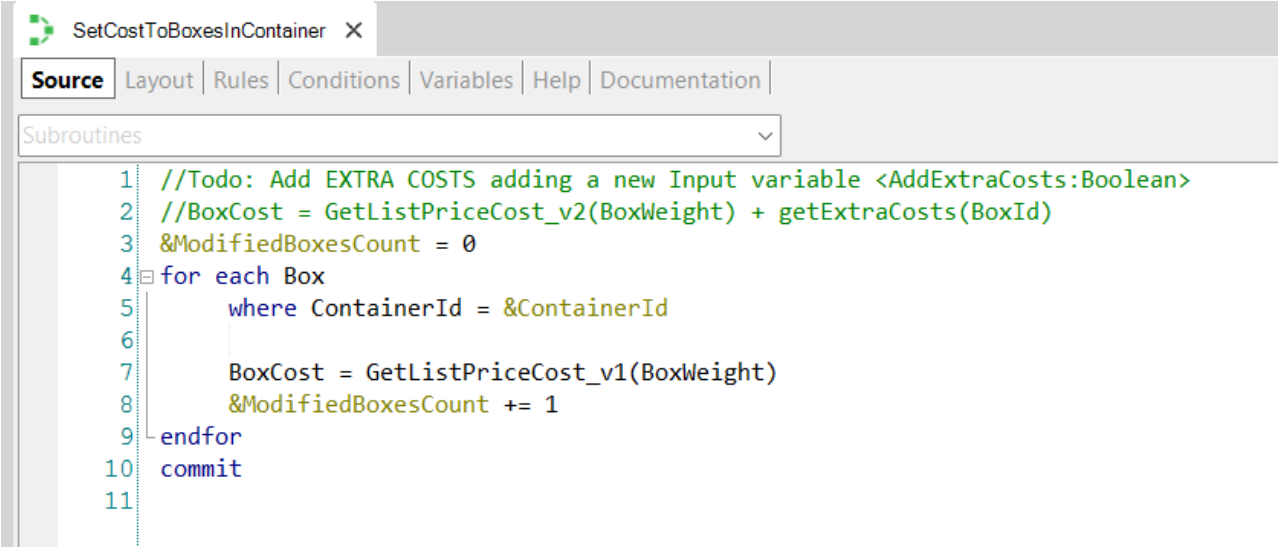
1) プロシージャーを右クリックして”開く”を選択、もしくはダブルクリックして開きます。



2) “Rules”タブをクリックします。このプロシージャーでは、入力として”&ContainerId”（コストを設定するボックスのコンテナ）を受け取り、出力として”&ModifiedBoxesCount”（変更されたボックスの数）を返します。

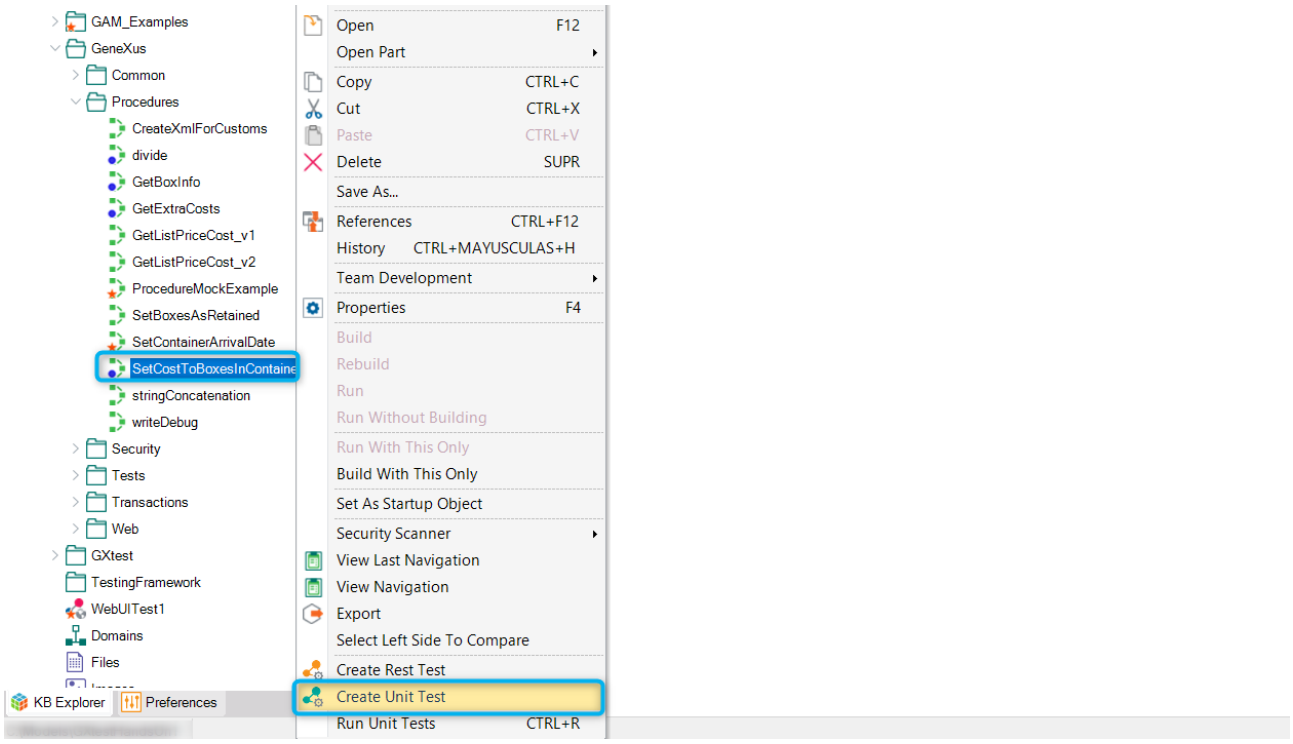


3) “Source”タブをクリックします。ご覧のように、プロシージャーのコードは、入力パラメータである”&ContainerId”と ID が一致するコンテナの各ボックスにコストを割り当てています。



プロシージャーからユニットテストを作成する

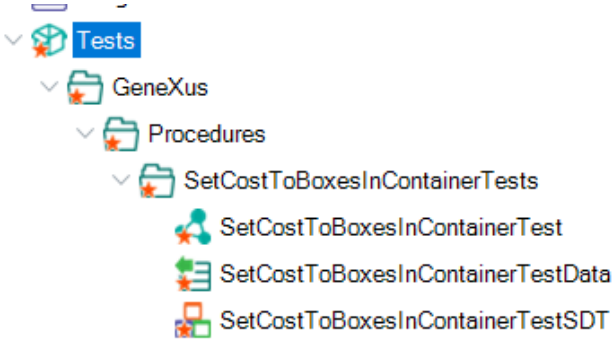
1) KB エクスプローラ画面で”SetCostToBoxesInContainer” プロシージャーを選択し、右クリック、”ユニットテストを作成”を選択します。



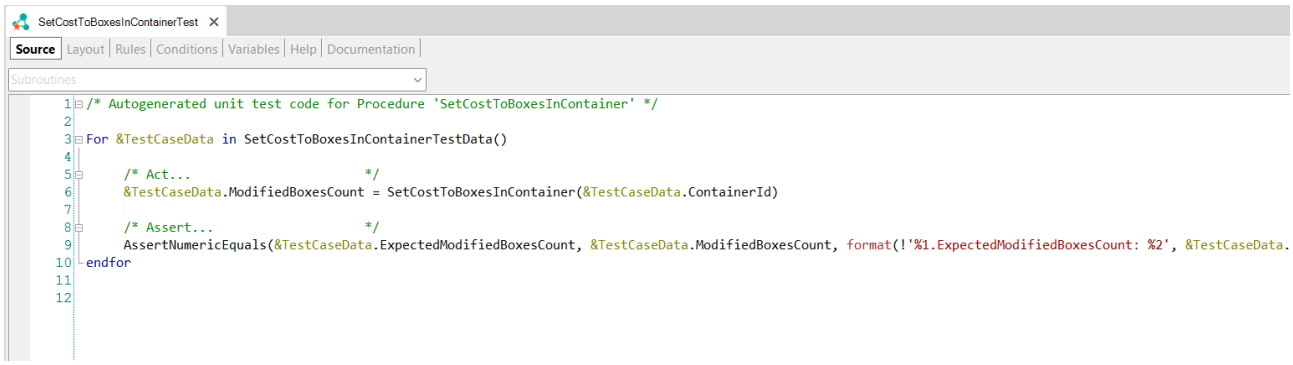
2) 以下のテストオブジェクトが生成されるまでお待ちください。

“SetCostToBoxesInContainerTest”, “SetCostToBoxesInContainerTestData”, “SetCostToBoxesInContainerTestSDT”

これらのオブジェクトは"Tests"モジュール内の"SetCostToBoxesInContainerTests"フォルダに表示されます。

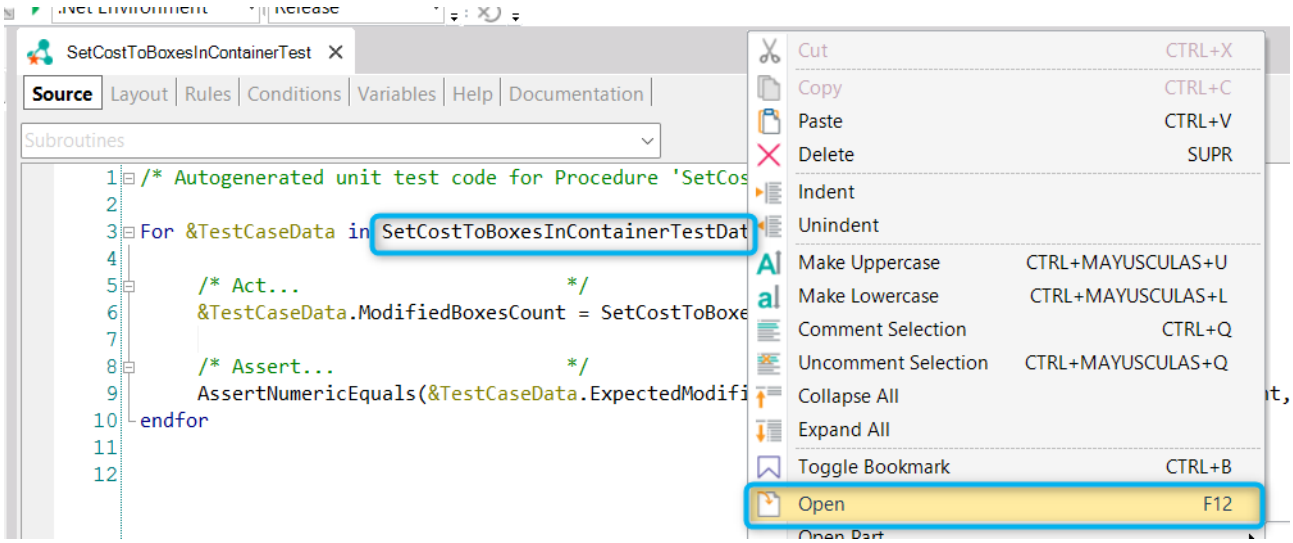


3) ご覧のように、GXtest では、テストが実行されるデータセットを含む "SetCostToBoxesInContainerTestData"データプロバイダを反復処理することにより、GeneXus コードテンプレートが自動生成されています。



ユニットテストパス

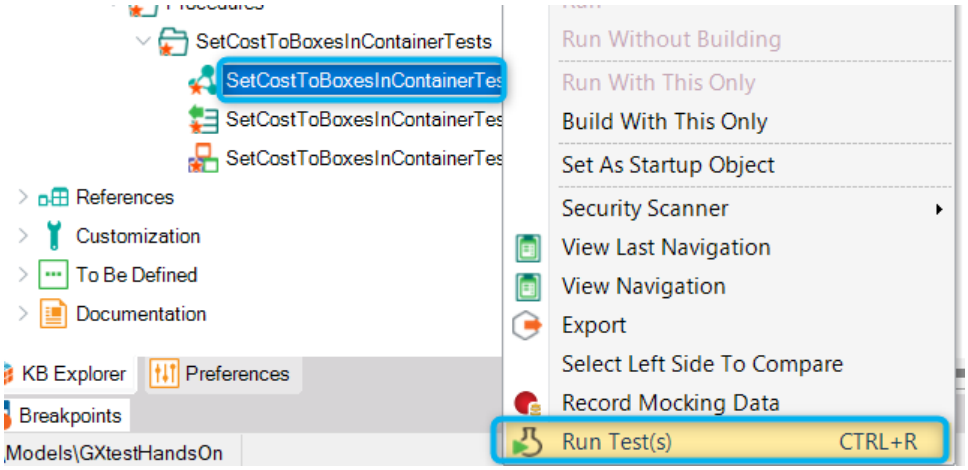
1) "SetCostToBoxesInContainerTestData"オブジェクトを右クリックし、「開く」を選択します。



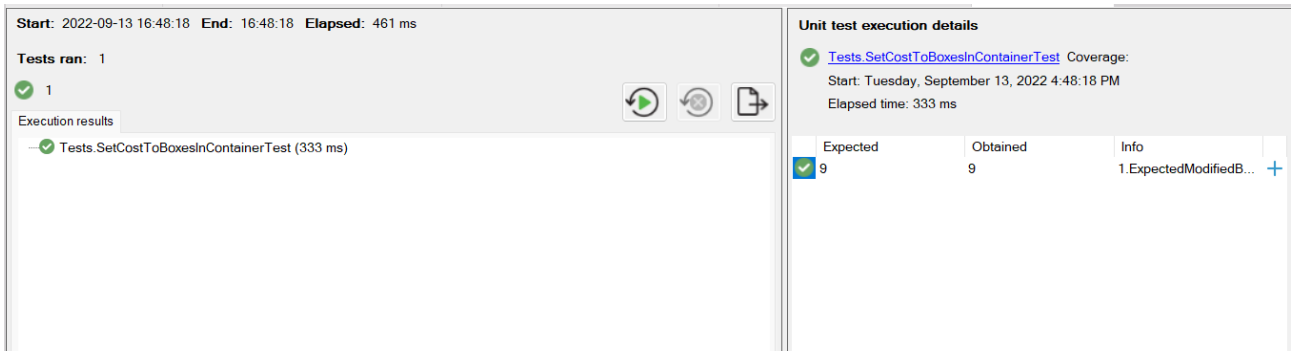
2) “SetCostToBoxesInContainerTestData”オブジェクトに下記のようにコードを記述します：

```
SetCostToBoxesInContainerTestSDT
{
        TestCaseId = '1'
        ContainerId = 1
        ExpectedModifiedBoxesCount = 9
        MsgModifiedBoxesCount = '9'
}
```

3) 変更を保存し、Unit Test を右クリックし、「テストを実行」を選択します。



4) テストの実行が終了すると、その結果が「テスト結果」に表示されます。
ご覧のように、修正されたボックスの予想数と取得された数が同じですので、テストは合格しています。



ユニットテストの失敗

失敗が予想されるテストを実行するには、データプロバイダの値を変更して、予想される数値と返される数値結果「9」が異なるようにする必要があります。

"SetCostToBoxesInContainerTestData"オブジェクトで編集する行を確認し、数値を変更し、「テストを実行」を選択してテストを実行します。

テストが失敗すると、得られた結果と期待される結果との差異が表示されます。

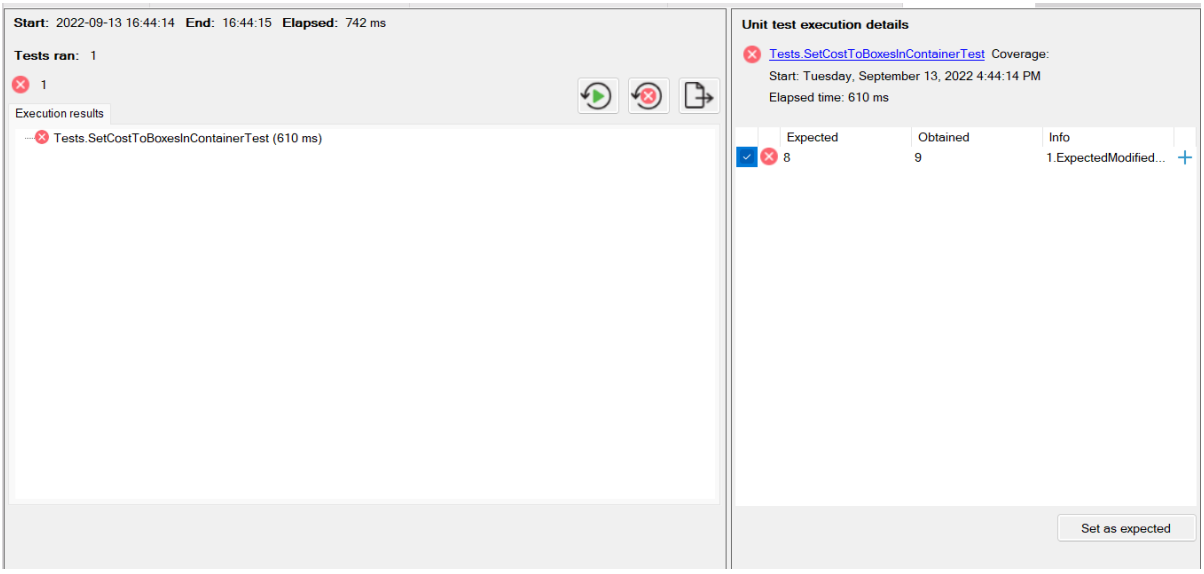
実行例：

"SetCostToBoxesInContainerTestData"オブジェクトに下記のようにコードを記述します：

```
SetCostToBoxesInContainerTestSDT
{
        TestCaseId = '1'
        ContainerId = 1
        ExpectedModifiedBoxesCount = 8
        MsgModifiedBoxesCount = '8'
}
```

1）変更を保存し、「テストを実行」を選択してテストを実行します。

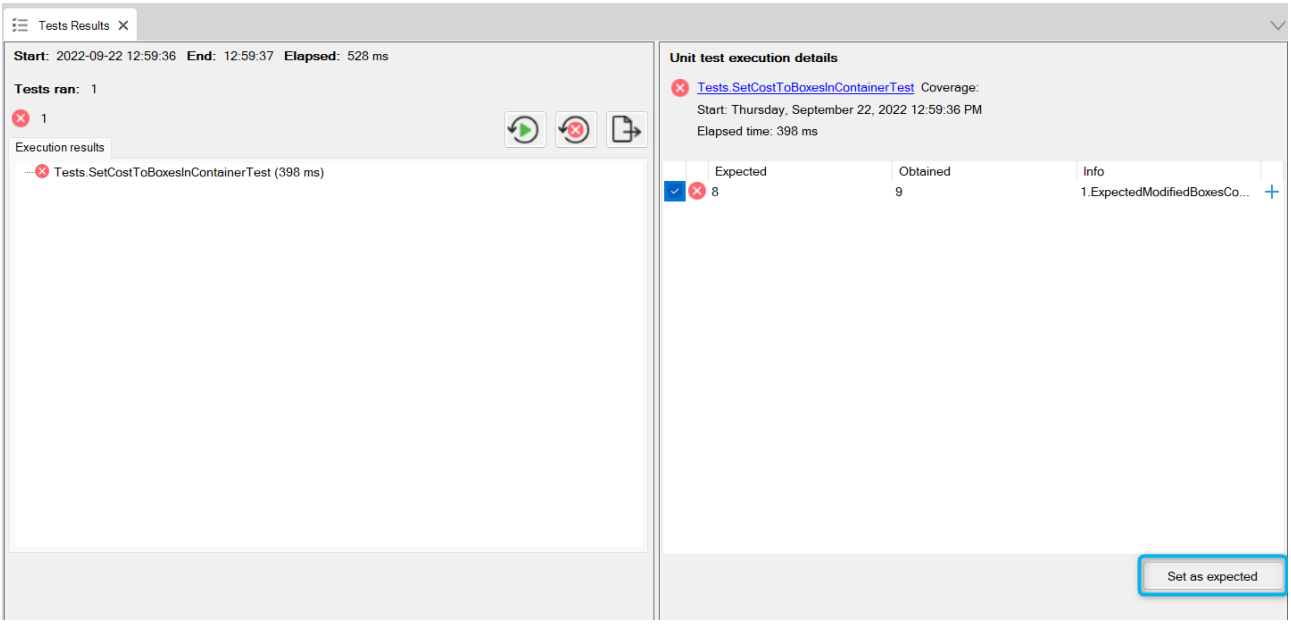
2) テストが終了すると、「テスト結果」に結果が表示され、エラー内容や期待した結果と得られた結果の差が表示されることを確認してください。



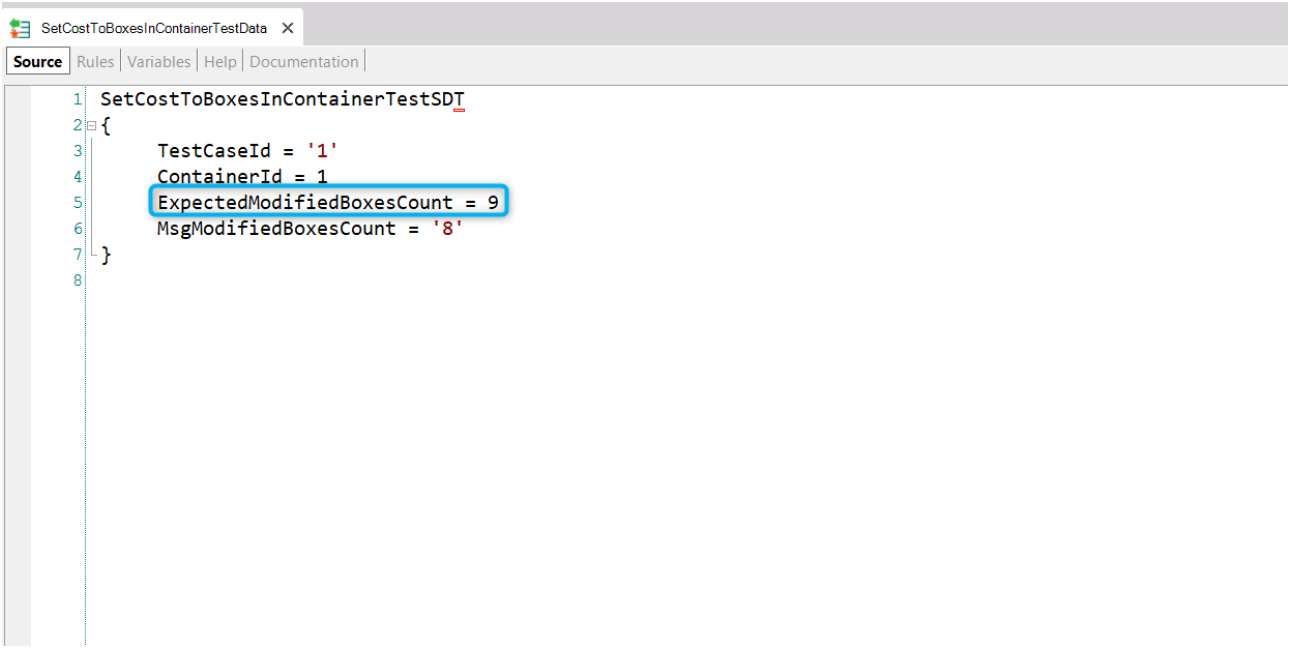
期待値として設定

実行結果が正しいかどうか常に確認するために、実行結果で期待される結果を設定する機能があります。「期待値として設定」ボタンをクリックすると、データプロバイダに期待する値が設定されます。この機能を理解するために、最後の例で試してみましょう。

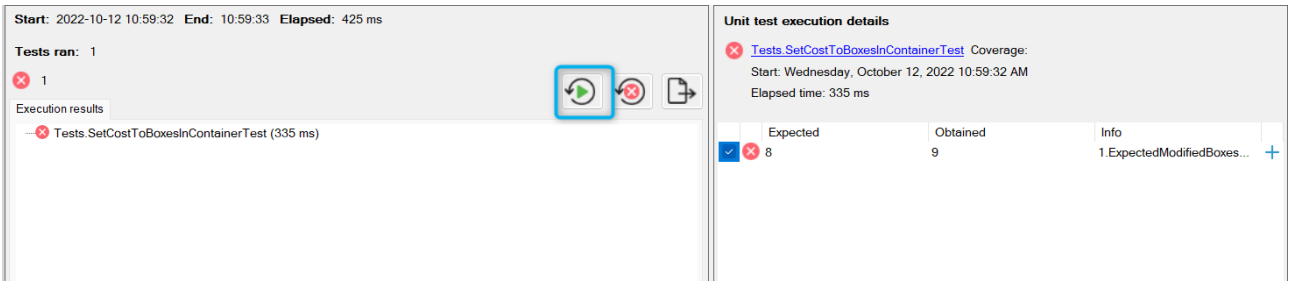
1) 「テスト結果」の「期待値として設定」をクリックします。



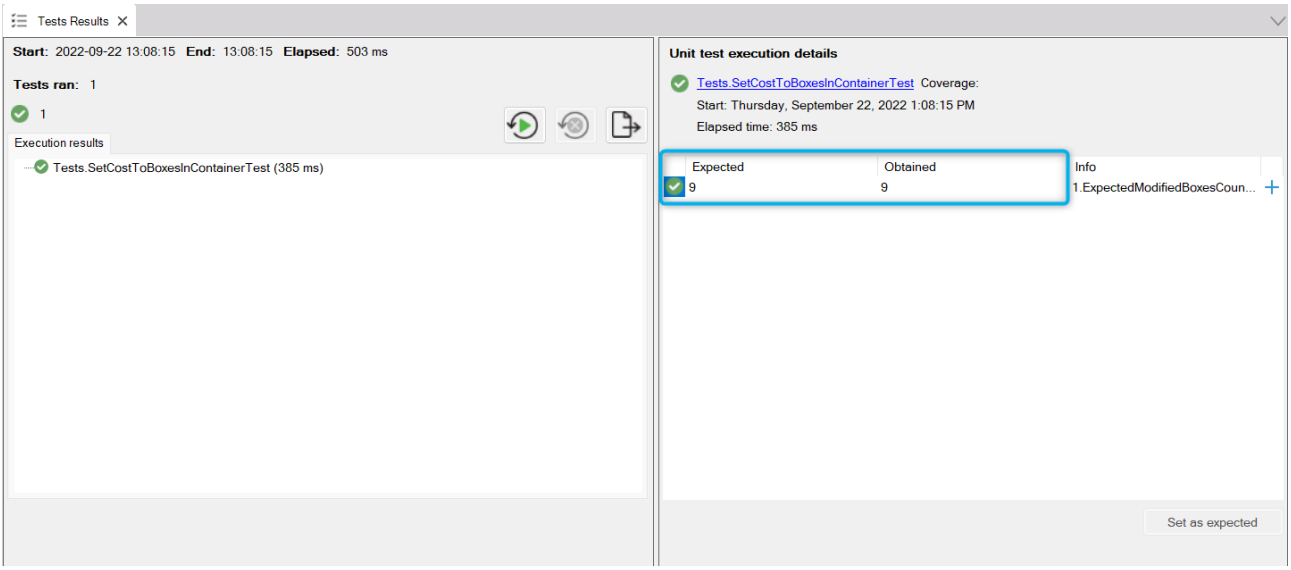
2) “SetCostToBoxesInContainerTestData"オブジェクトを開き、"ExpectedModifiedBoxesCount"の数値が”9”に変化していることを確認します。



3) 「再実行」をクリックして、"SetCostToBoxesInContainerTest"を再度実行してください。



4) ご覧の通り、テストは成功し、期待値として設定された値と同じ結果を得ることができました。



この機能は、テストケースの期待値を設定するために、初回のテスト実行（例えば、Expected result を 0 または Null に設定する）で使用することに注意してください。前に示した例は、これをどのように動作させるかを示すだけのものです。

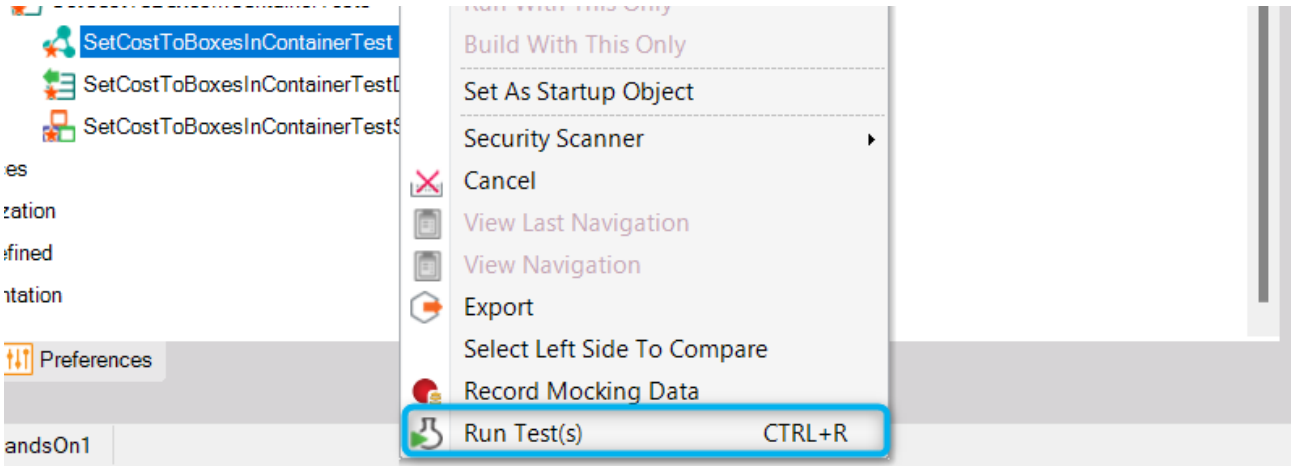
データベース状態の検証

“SetCostToBoxesInContainer”プロシージャーが実行されたときに、実際にデータベースの状態が変化しているかどうかを検証します。これを行うには、データベースの状態をチェックするためのアサーションを追加します。この場合、コンテナの各ボックスのコストが 0 より大きいかどうかを検証します。

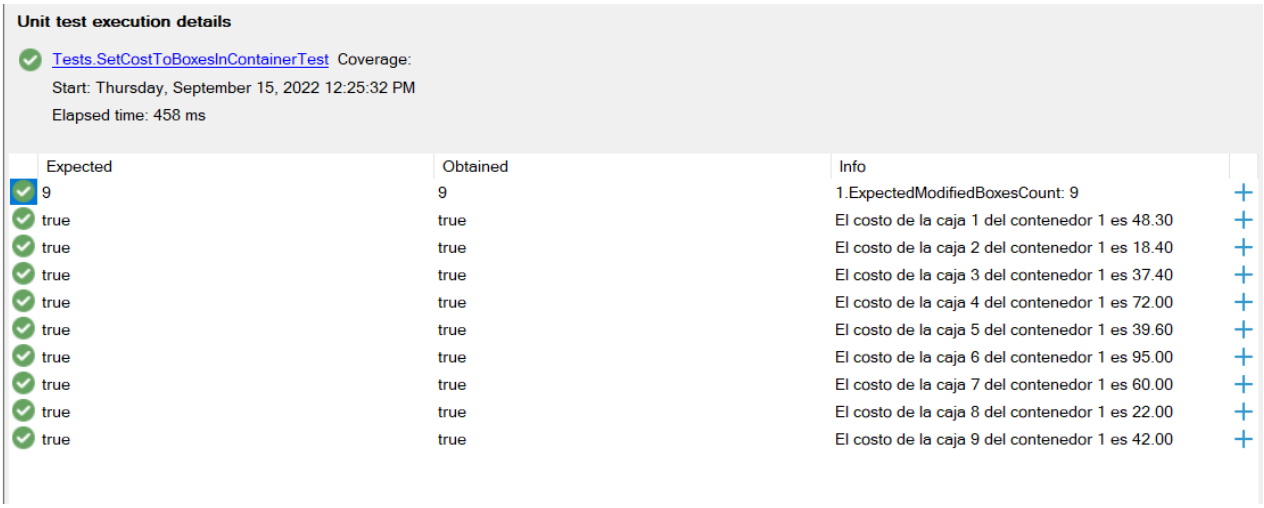
1) "SetCostToBoxesInContainerTest"オブジェクトを開きます。
2) 既存のアサーションに、以下のアサーションを追加する。

```
/* Assertion of database status. */

for each Box

    where ContainerId = &TestCaseData.ContainerId

AssertBoolEquals(true, BoxCost > 0, Format("El costo de la caja %1 del contenedor %2 es %3",BoxId, ContainerId, BoxCost))

endfor
```

3) 変更を保存し、Unit Test オブジェクトを右クリックし、「テストを実行」を選択してテストを実行します。



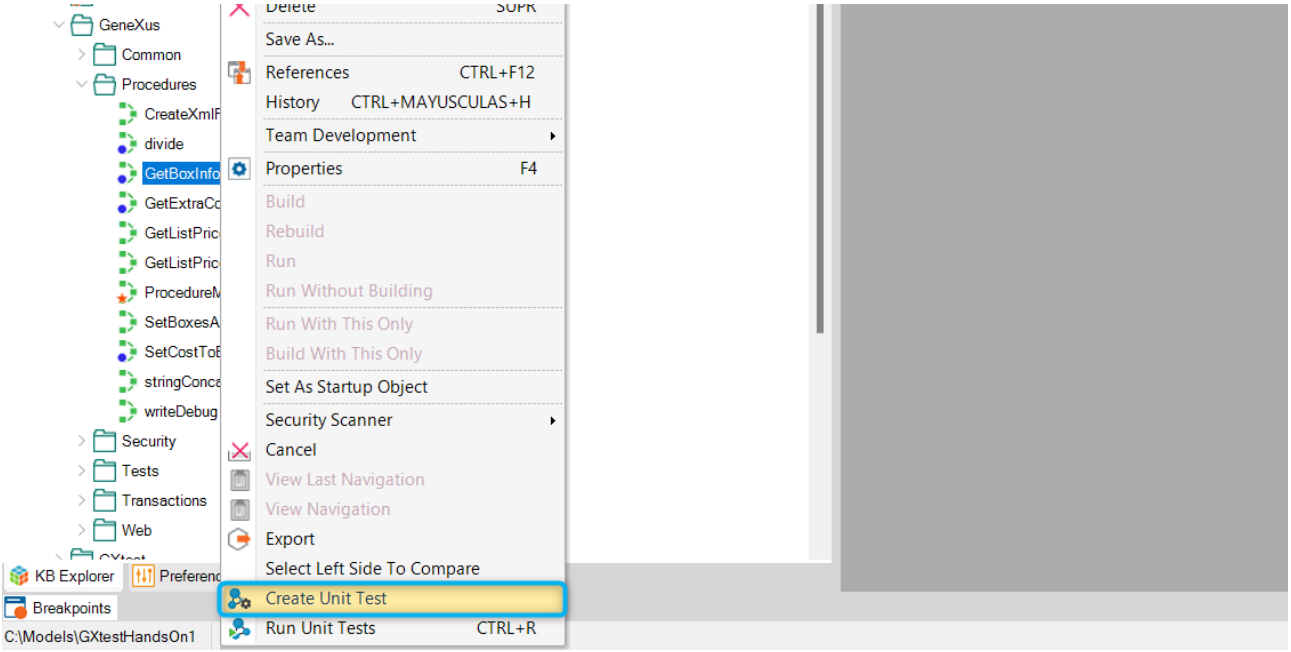
結果画面では、1 行目にプロシージャーの出力パラメータの検証、残りの行に各ボックスのコストのデータベースステータスの検証結果が表示されます。



データベースモッキングによる実行

データベースモッキング機能を理解するために、パラメータで受け取った”BoxId”で特定されるボックスに関する情報を返す”GetBoxInfo”プロシージャーのテストを実行します。今回は、Container = 1 の BoxId = 1 の情報をチェックするモックを使ったテストケースを作成します。

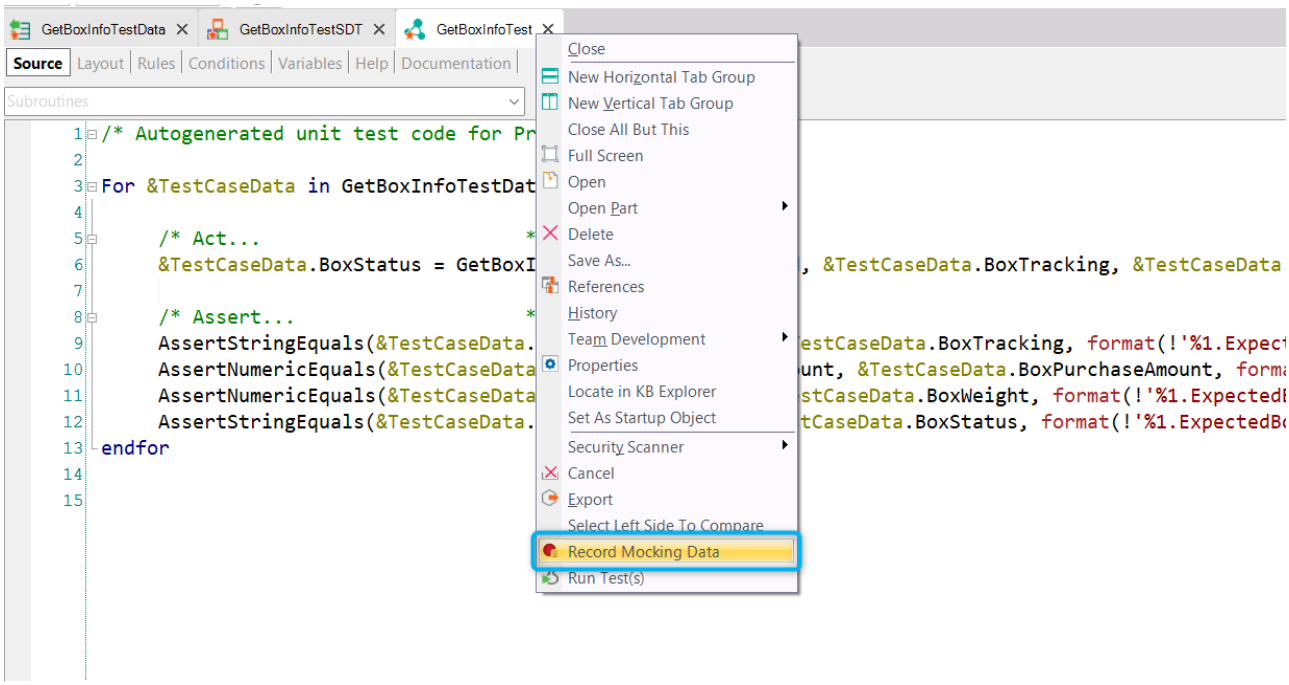
1) KB の”GetBoxInfo”プロシージャーを右クリックし、「ユニットテストを作成」を選択して、ユニットテストを作成します。



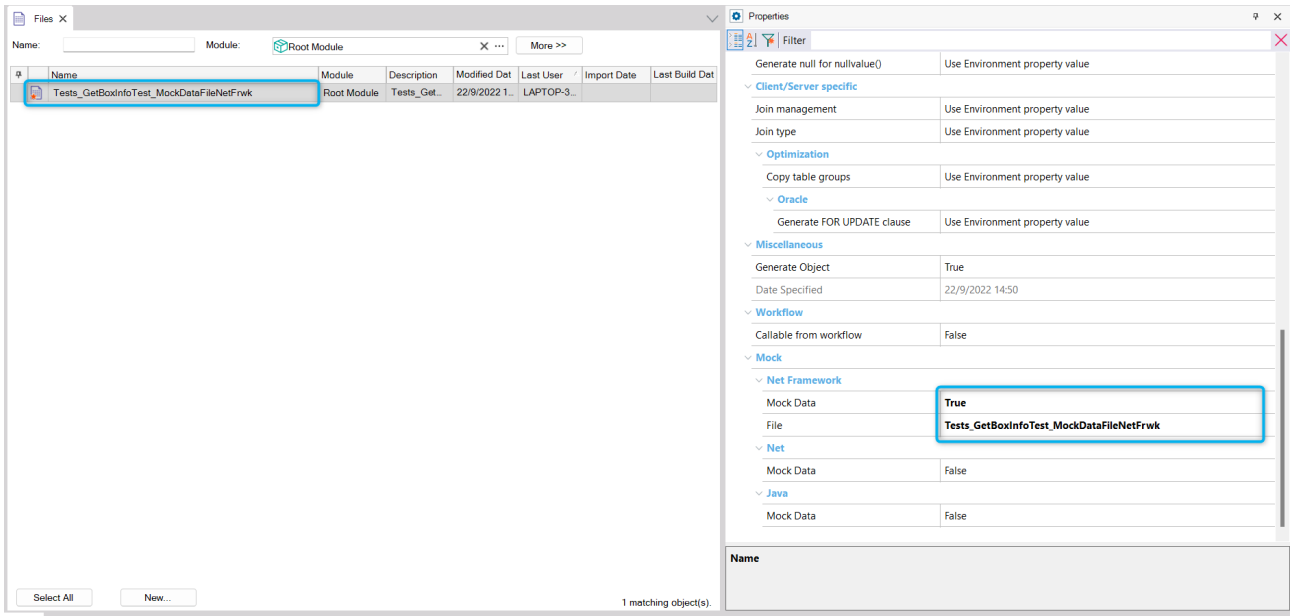
2) "GetBoxInfoTestData"オブジェクトを開き、”BoxId=1”のデータに対するアサーションを実施するよう下記のようにコードの記載を変更します：

```
GetBoxInfoTestSDT
{
        TestCaseId = '1'
        BoxId = 1
        ExpectedBoxTracking = '1146HFGDS3'
        MsgBoxTracking = '1146HFGDS3'
        ExpectedBoxPurchaseAmount = 180
        MsgBoxPurchaseAmount = '180'
        ExpectedBoxWeight = 2.3
        MsgBoxWeight = '2.3'
        ExpectedBoxStatus = Status.ATDESTINATION
        MsgBoxStatus = 'Status.ATDESTINATION'
        ExpectedBoxOriginArrivalDate = #2018-10-22#
        MsgBoxOriginArrivalDate = '#2018-10-22#'
        ExpectedBoxCost = 48.3
        MsgBoxCost = '48.3'
}
```

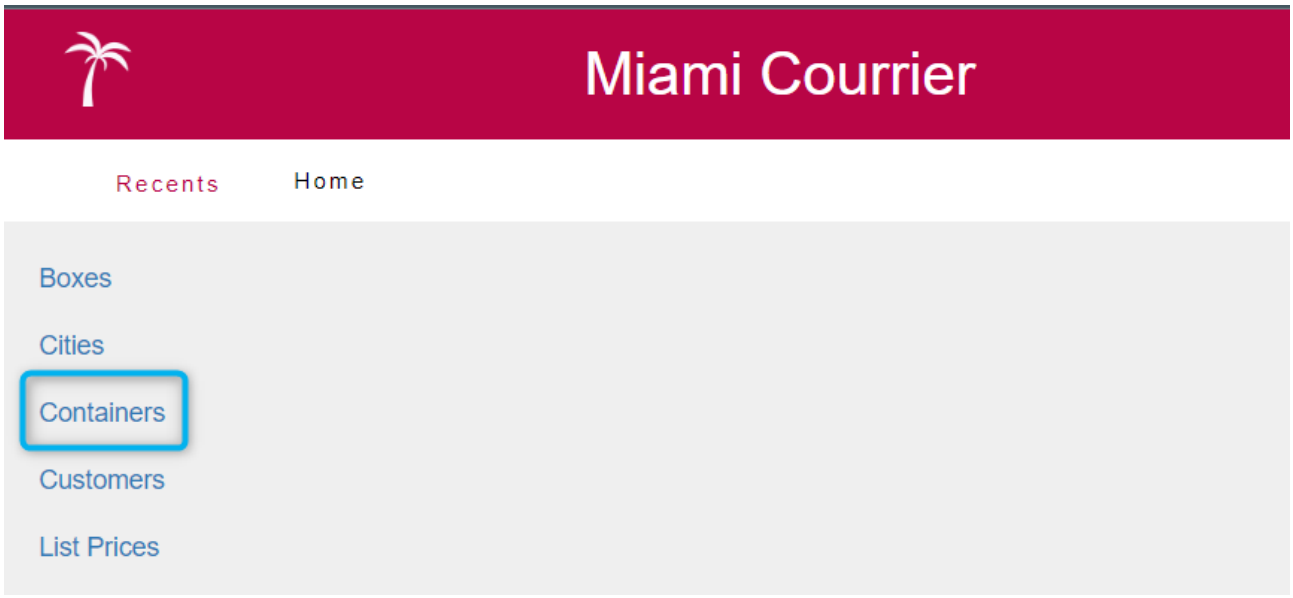
3) “GetBoxInfoTestData"オブジェクトの変更内容を保存し、Unit Test を右クリックし、「モックデータを記録」を選択します。

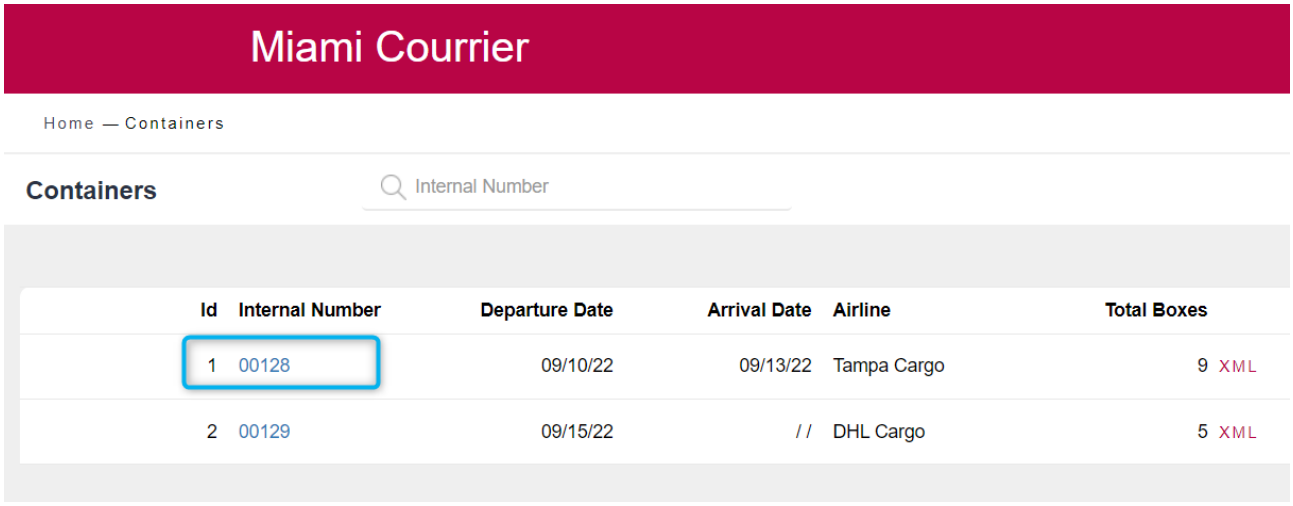


このように、”Files”画面上に”Tests_GetBoxInfoTest_MockDataFileNetFrwk”という名前でファイルが作成されます。” GetBoxInfoTest”オブジェクトの”Mock Data”プロパティが”True”に変わり、作成されたファイルが自動的に”File”プロパティに割り当てられます。



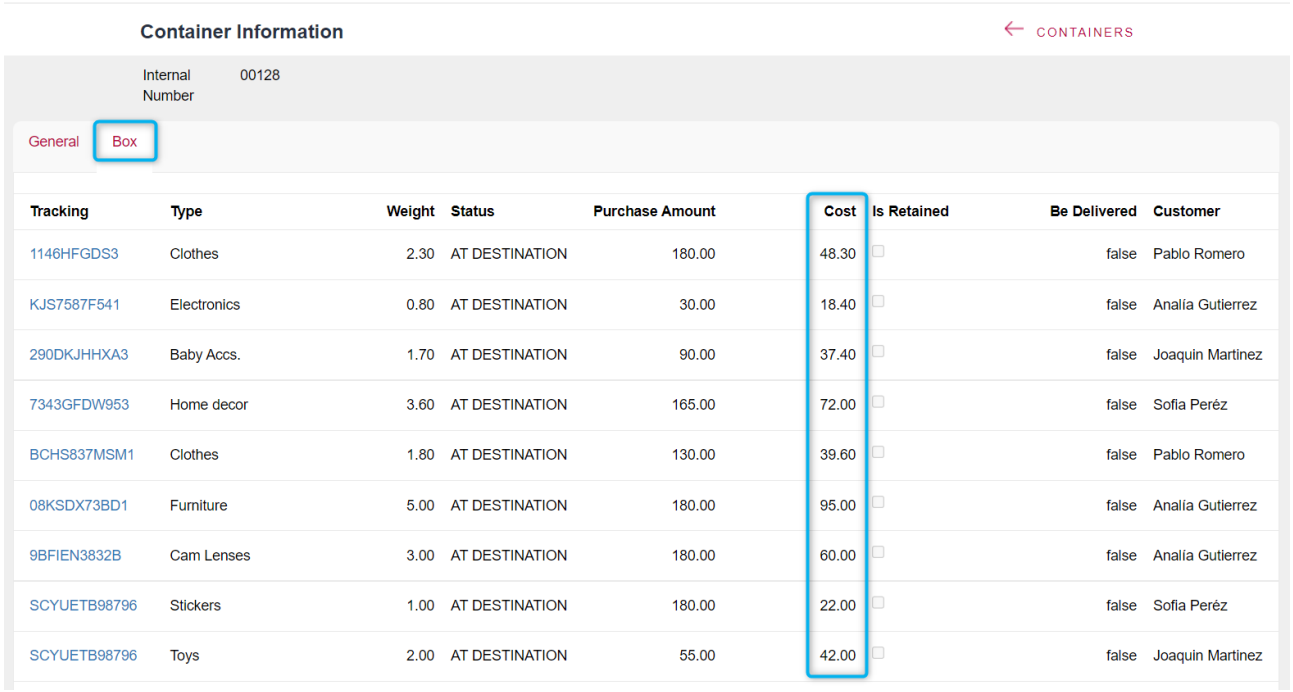
4）実行したアプリケーションで「Containers」をクリックします。ID1 のコンテナにてリンクである「00128」をクリックし、コンテナ情報にアクセスします。





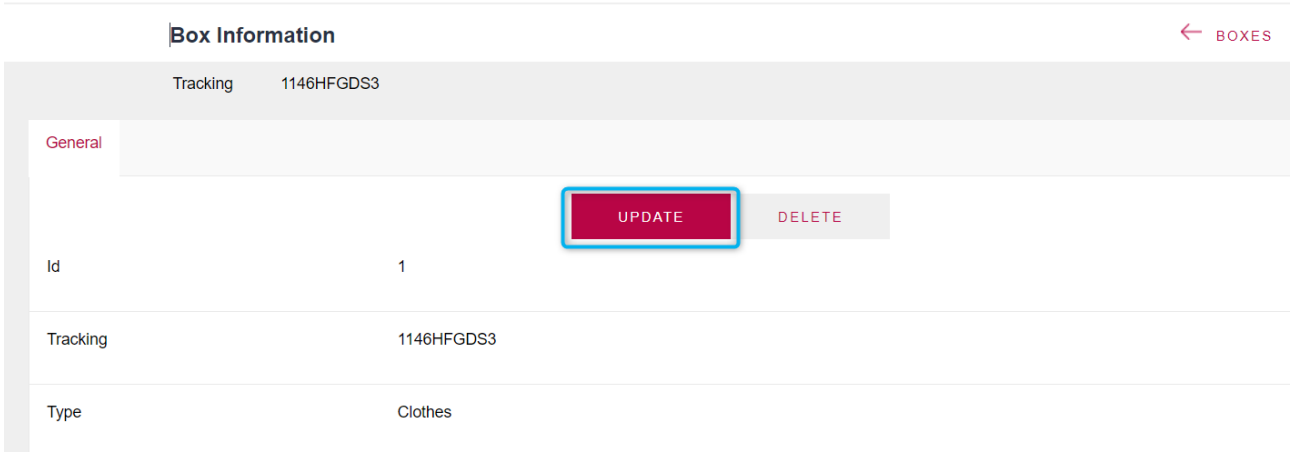
5) 「Box」タブをクリックします。

すると、”SetCostToBoxesInContainer”プロシージャーで設定した数値が各ボックスの「Cost」の欄に表示されていることがわかります。

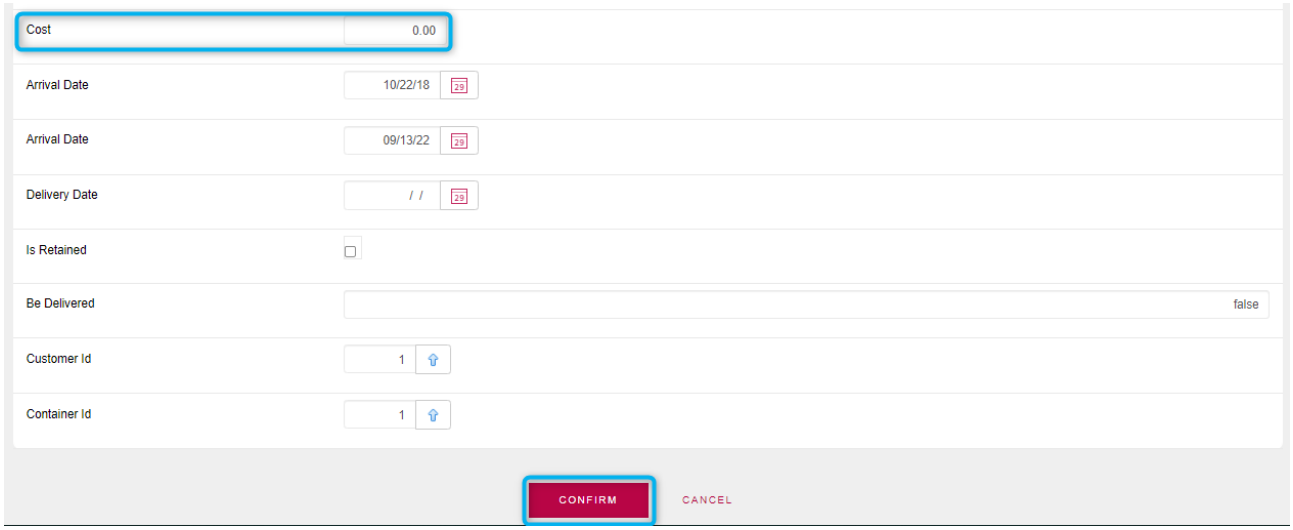


Tracking	Type	Weight	Status	Purchase Amount	Cost	Is Retained	Be Delivered	Customer
1146HFGDS3	Clothes	2.30	AT DESTINATION	180.00	48.30		false	Pablo Romero
KJS7587F541	Electronics	0.80	AT DESTINATION	30.00	18.40		false	Analía Gutierrez
290DKJHHXA3	Baby Accs.	1.70	AT DESTINATION	90.00	37.40		false	Joaquin Martinez
7343GFDW953	Home decor	3.60	AT DESTINATION	165.00	72.00		false	Sofia Peréz
BCHS837MSM1	Clothes	1.80	AT DESTINATION	130.00	39.60		false	Pablo Romero
08KSDX73BD1	Furniture	5.00	AT DESTINATION	180.00	95.00		false	Analía Gutierrez
9BFIEN3832B	Cam Lenses	3.00	AT DESTINATION	180.00	60.00		false	Analía Gutierrez
SCYUETB98796	Stickers	1.00	AT DESTINATION	180.00	22.00		false	Sofia Peréz
SCYUETB98796	Toys	2.00	AT DESTINATION	55.00	42.00		false	Joaquin Martinez

6) 「Box 1」のボックスに対するリンクである「1146HFGDS3」をクリックし、「UPDATE」をクリックします。

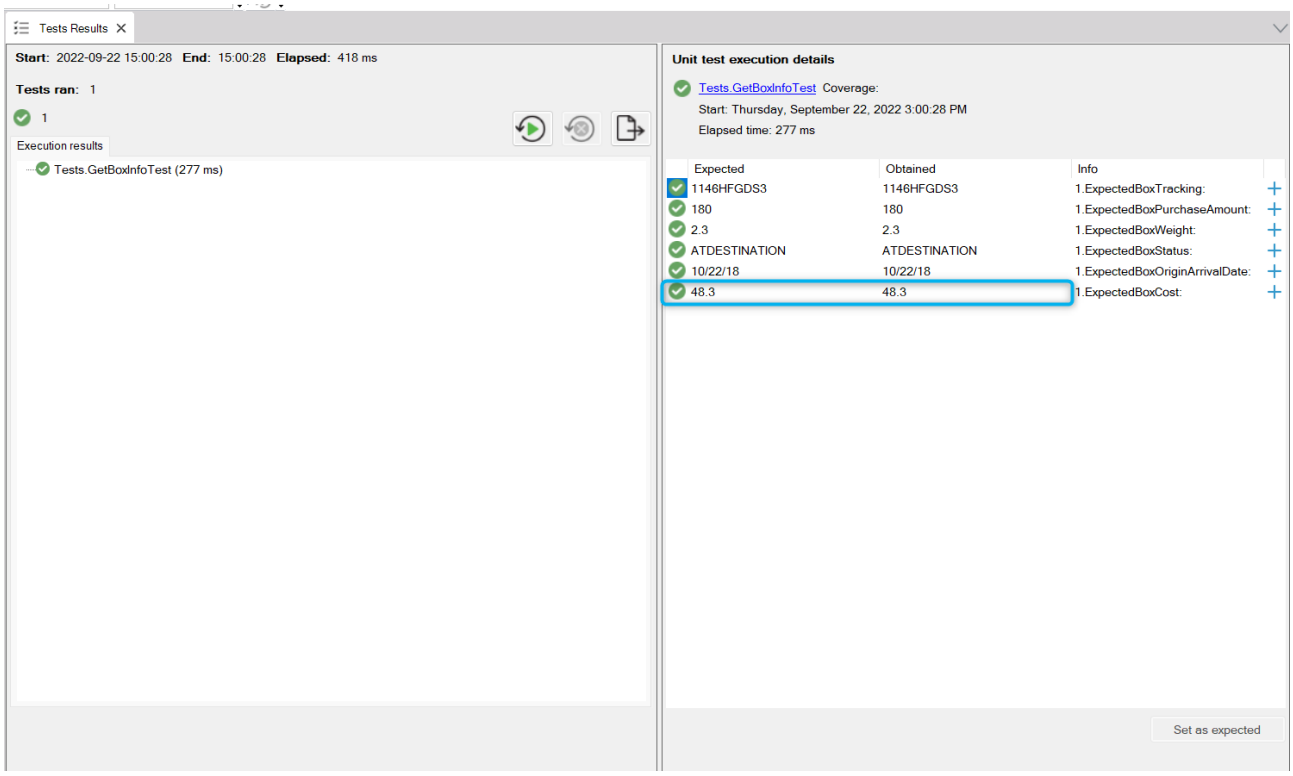


7) 「Cost」セルの値を”0.00”に変更し、「CONFIRM」をクリックします。



8) 「テスト結果」から「再実行」をクリックして、もう一度テストを実行します。
今度は、記録されたモッキングデータを使用してテストを実行します。

テストの実行が終了すると、「テスト結果」に結果が表示されます。コストの値である”48.3”は、先ほど編集したデータベースからではなく、モックファイルから取得したものであることがわかります。



動的データのためのモックファイル編集

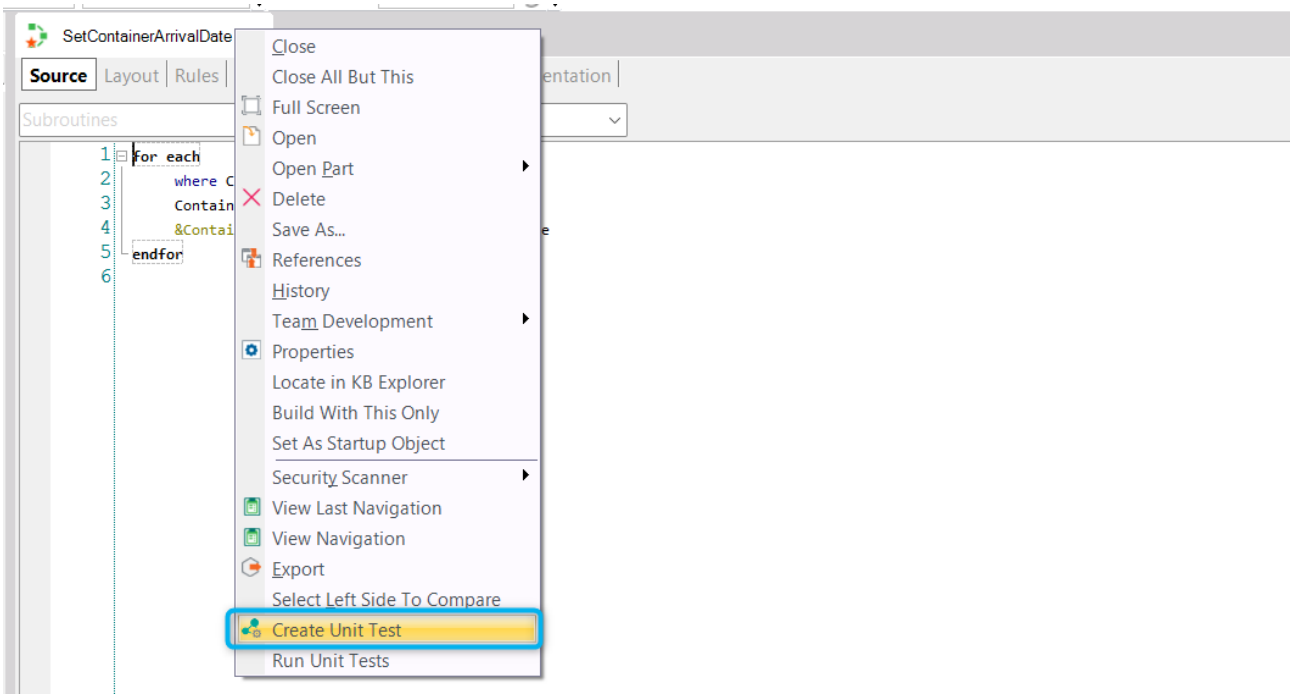
“SetContainerArrivalDate”プロシージャーでは、コンテナ 1 の到着日（&ContainerArrivalDate）が&Today と設定されていることが分かります。

このプロシージャーの Unit Test を、モックを記録した日とは別の日に実行すると、モックファイルに登録された日付と、システムがリアルタイムに取得した日付に差異が生じます。これがいわゆる”動的データ”です。

このようなことが起こらないようにするために、記録されたデータを修正し、すべての実行で機能する必要があります。これは、JSON ファイルの”KeyPattern”フィールドを変更し、動的な値を正規表現に置き換える必要があります。それでは、その方法を見てみましょう。

“SetContainerArrivalDate”プロシージャーに対するユニットテストを作成し、実行データをモック記録します。

1) “SetContainerArrivalDate”プロシージャーを右クリックし、「ユニットテストを作成」を選択します。

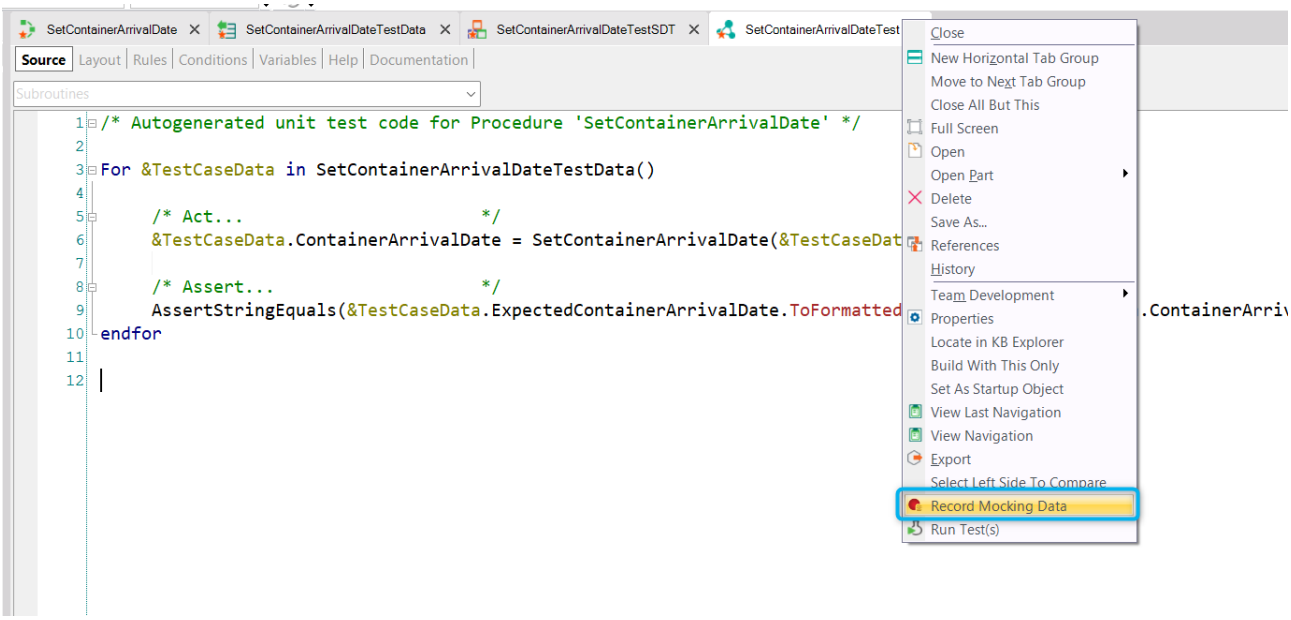


2) 以下のテストオブジェクトが生成されるまでお待ちください。
"SetContainerArrivalDateTest","SetContainerArrivalDateTestData",
"SetContainerArrivalDateTestSDT"

3) “SetContainerArrivalDateTestData”オブジェクトを開き、下記のようにコードの記載を変更します：

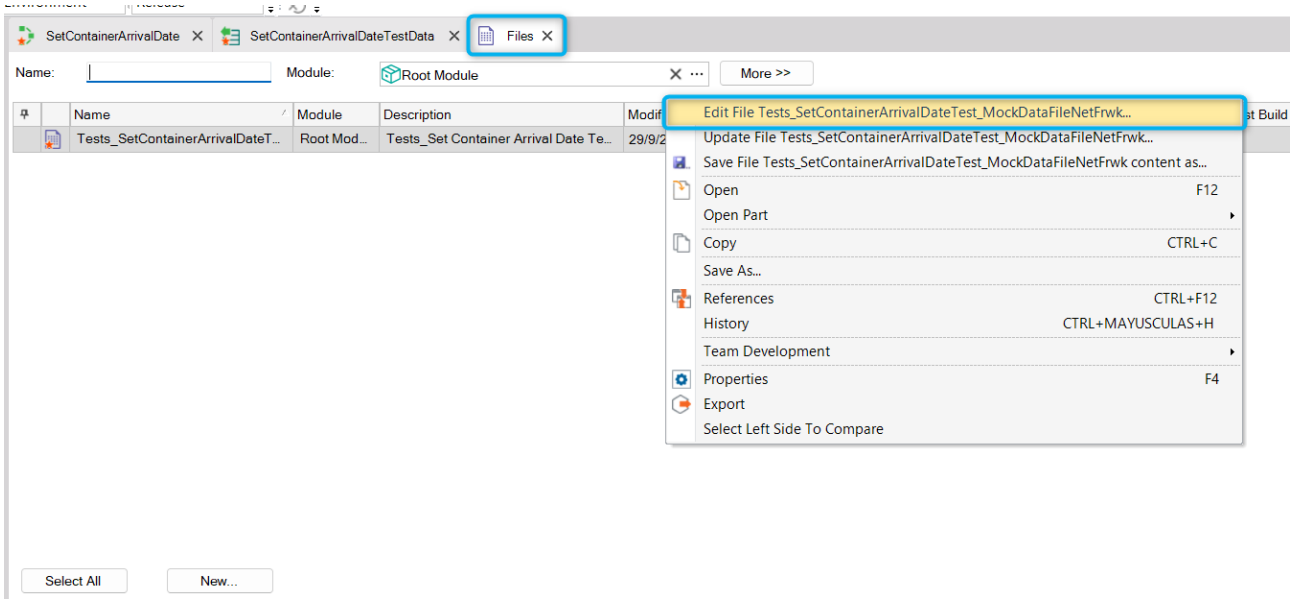
```
SetContainerArrivalDateTestSDT
{
        TestCaseId = '1'
        ContainerId = 1
        ExpectedContainerArrivalDate = #2023-01-01#
        MsgContainerArrivalDate = ''
}
```

4) "SetContainerArrivalDateTest"を右クリックし、「モックデータを記録」を選択します。



ここまでは、”SetContainerArrivalDate”プロシージャーをテストするユニットテストを作成し、その実行データを現在のシステム日付で記録してきました。続いて、受け取った変数の日付に関係なくテストが正常に実行されるようするため、モックファイルを編集してみましょう。

1) “Files”を開き、先ほどの操作で作成された”Tests_SetContainerArrivalDateTest_MockDataFileNetFrwk”という名前のファイルを右クリックし、「ファイルの編集」を選択します。
このとき、”.gxtest”というファイルを開くためのダイアログが出ますので、テキストファイルを編集可能なエディターを選択します。



"Key"フィールドの引用符で囲まれた内容をコピーします。

2) "KeyPattern"フィールドの引用符の中に貼り付けます。

3) “KeyPattern”フィールドに張り付けた内容からコンテナ 1 の到着日として編集する値を特定します。
4) 日付の正規表現である「“¥¥d{2,4}/¥¥d{1,2}/¥¥d{1,2}”」を入力して、その値を編集します。これは正確な文字列を期待するものではなく、システムが受け取った日付と一致するかを確認するものです。
5) 変更を保存し、「テスト結果」から「再実行」を選択してテストを実行します。

テストが実行されると、関連するモックファイルデータがロードされます。
"ContainerArrivalDate"の値は、実行時に記述された文字列ではなく、正規表現に基づく値が利用されます。

実行例:

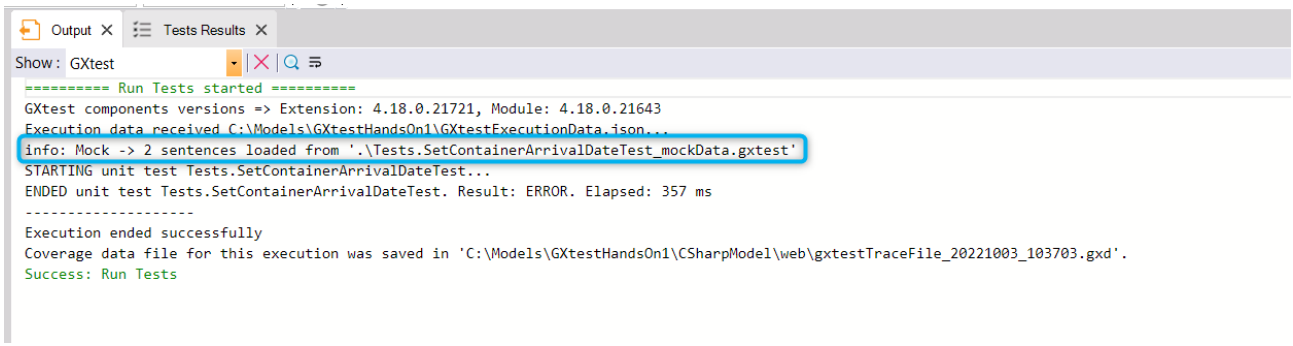
1) 「Key Pattern」行の値「“2023¥/01¥/01”」を正規表現である「{{¥¥d{2,4}/¥¥d{1,2}/¥¥d{1,2}}}」に置き換えます。

“Key Pattern”のコードは以下の通りです:

"{{¥¥d{2,4}/¥¥d{1,2}/¥¥d{1,2}}} 0:00:00,1,True,UPDATE [Container] SET[ContainerArrivalDate]=@ContainerArrivalDate WHERE [ContainerId] = @ContainerId"

2) 変更を保存し、「テスト結果」から「再実行」を選択してテストを実行します。

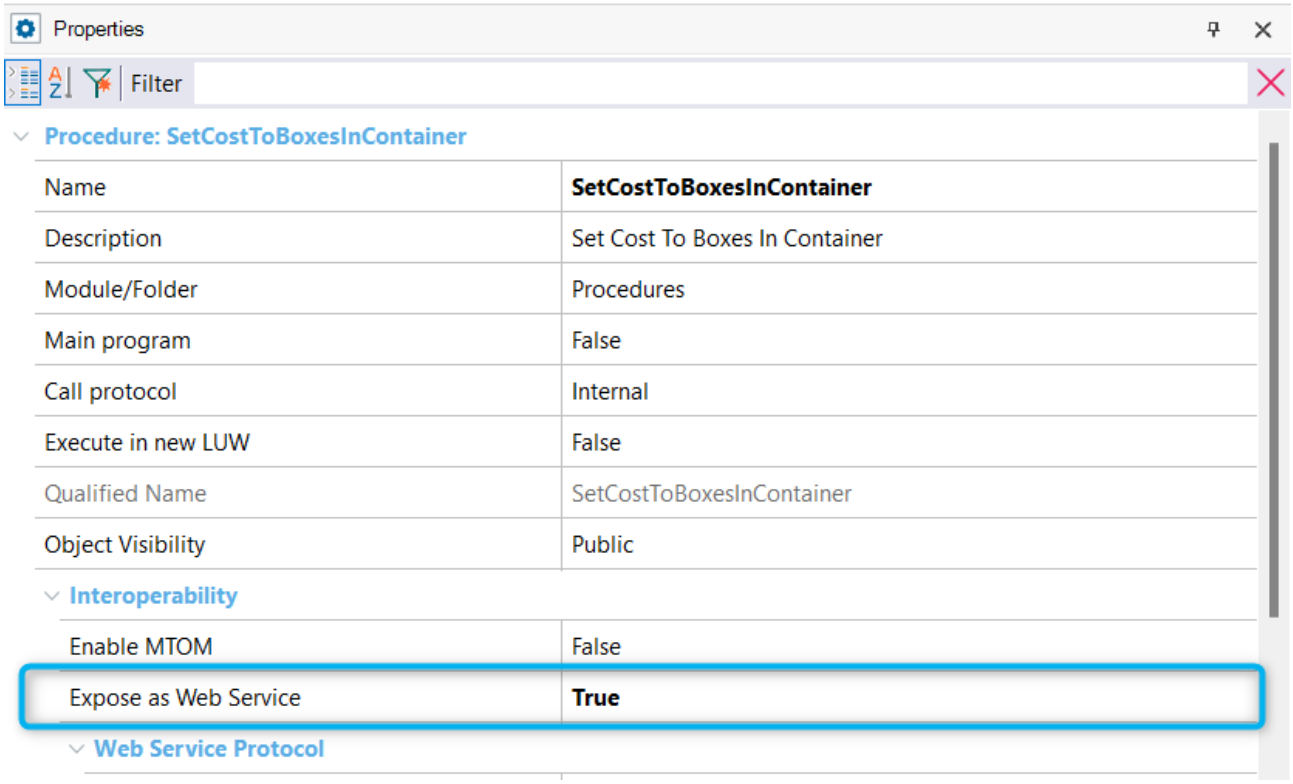
3) 出力画面では、JSON ファイルから読み込んだデータで実行が成功したことを確認してください。



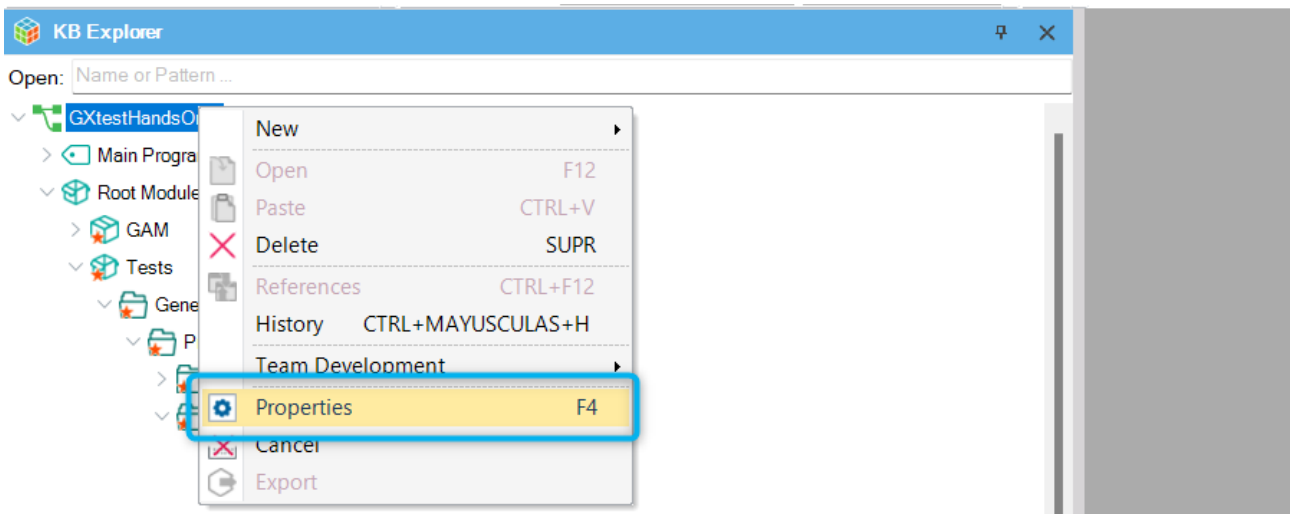
Rest テスト

REST サービスとして実装されたプロシージャーは HTTP プロトコル経由で実行してテストすることが可能です。このセクションでは、GAM 認証を有効にした"SetCostToBoxesInContainer"プロシージャーの REST サービステストを作成する方法について順を追って説明します。

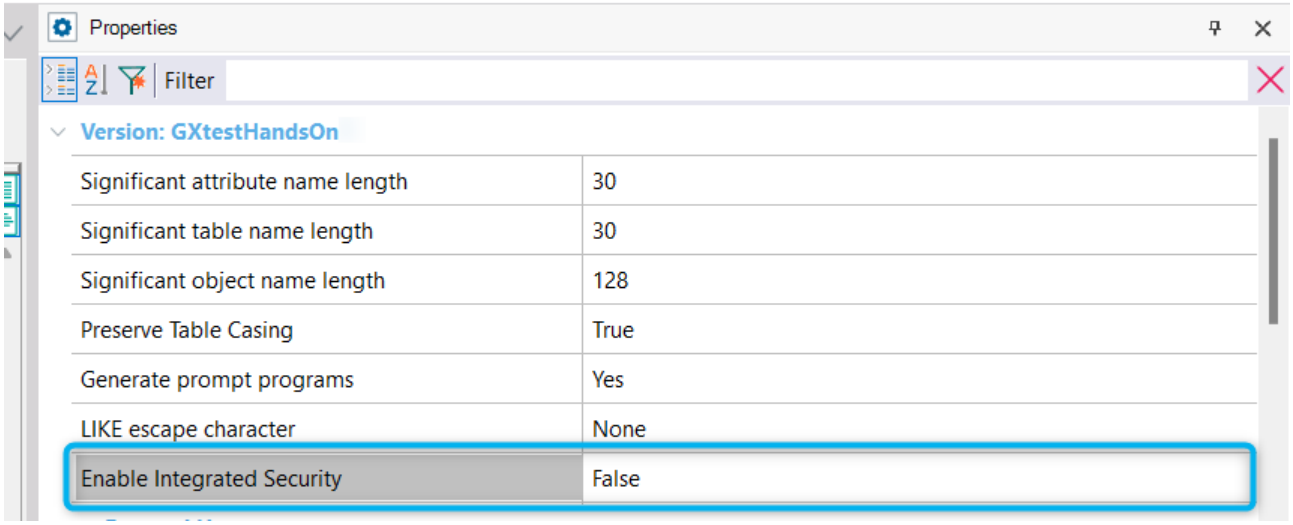
1) "SetCostToBoxesInContainer"プロシージャーのプロパティを開き、"Expose as Web Service"プロパティを"True"に変更します。



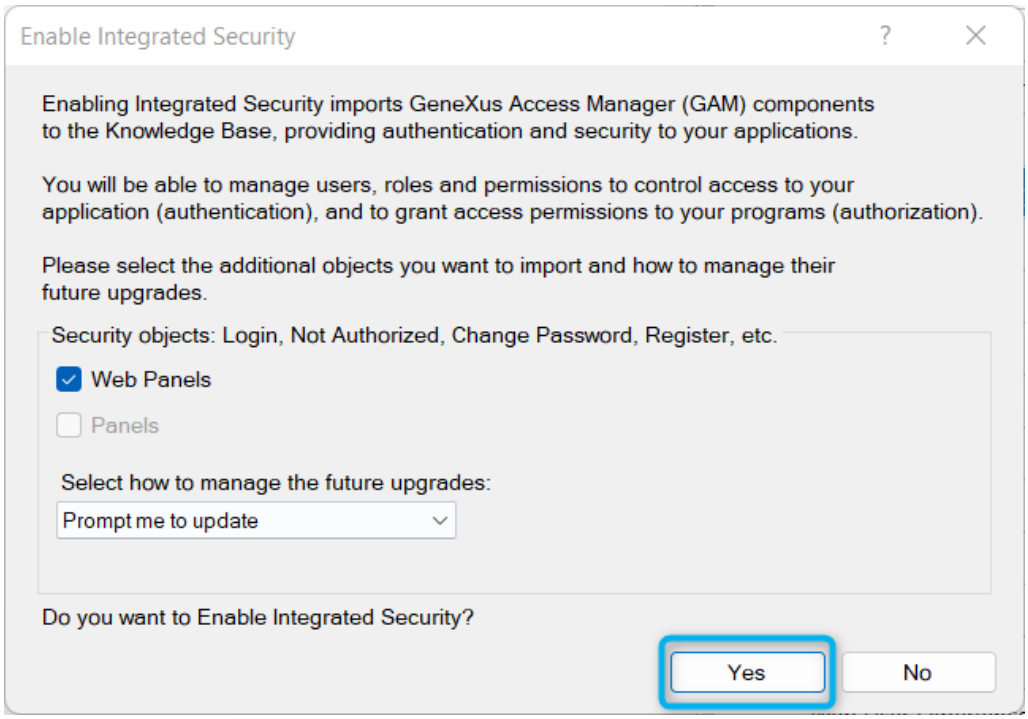
2) 「設定」にて最上位の階層を右クリックし「プロパティ」を選択し、KB のプロパティを開きます。



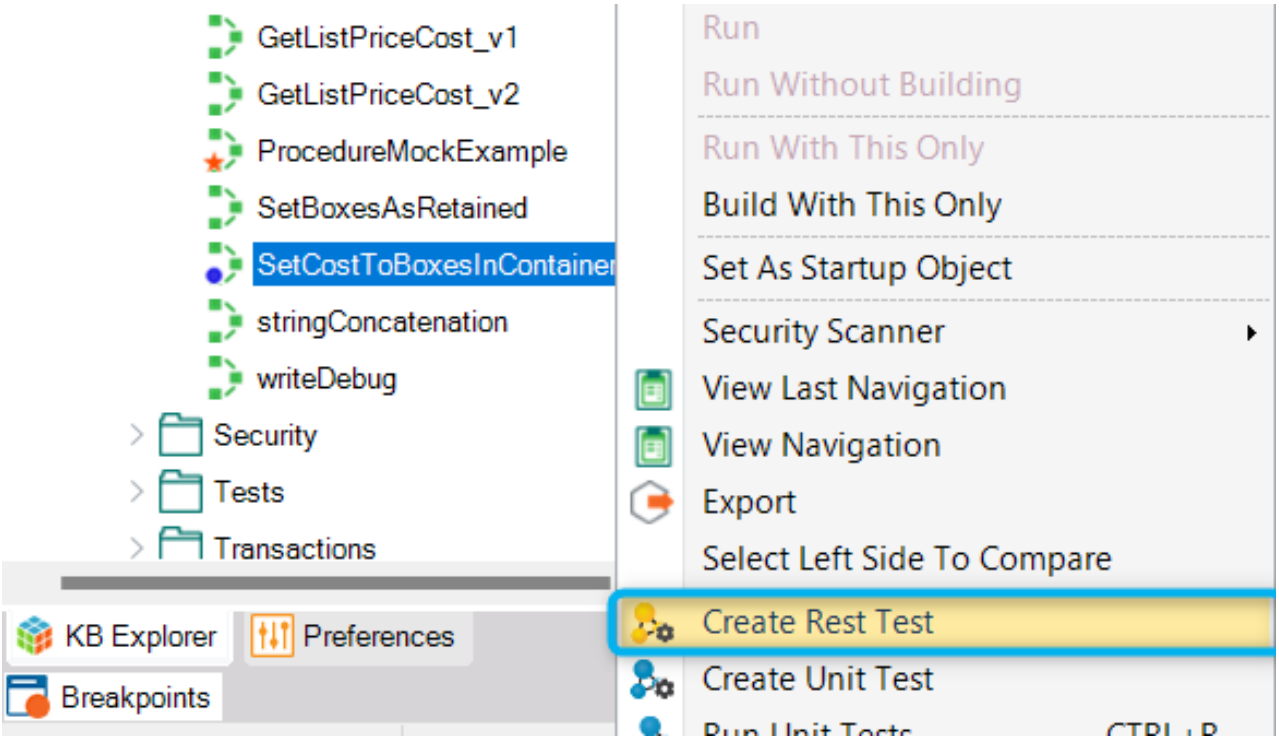
3) "Enable Integrated Security"プロパティを確認します。



4) "Enable Integrated Security"プロパティを"True"に変更します。ポップアップウィンドウが GAM コンポーネントのインポートを確認するよう要求しますので、「インストール」をクリックします。



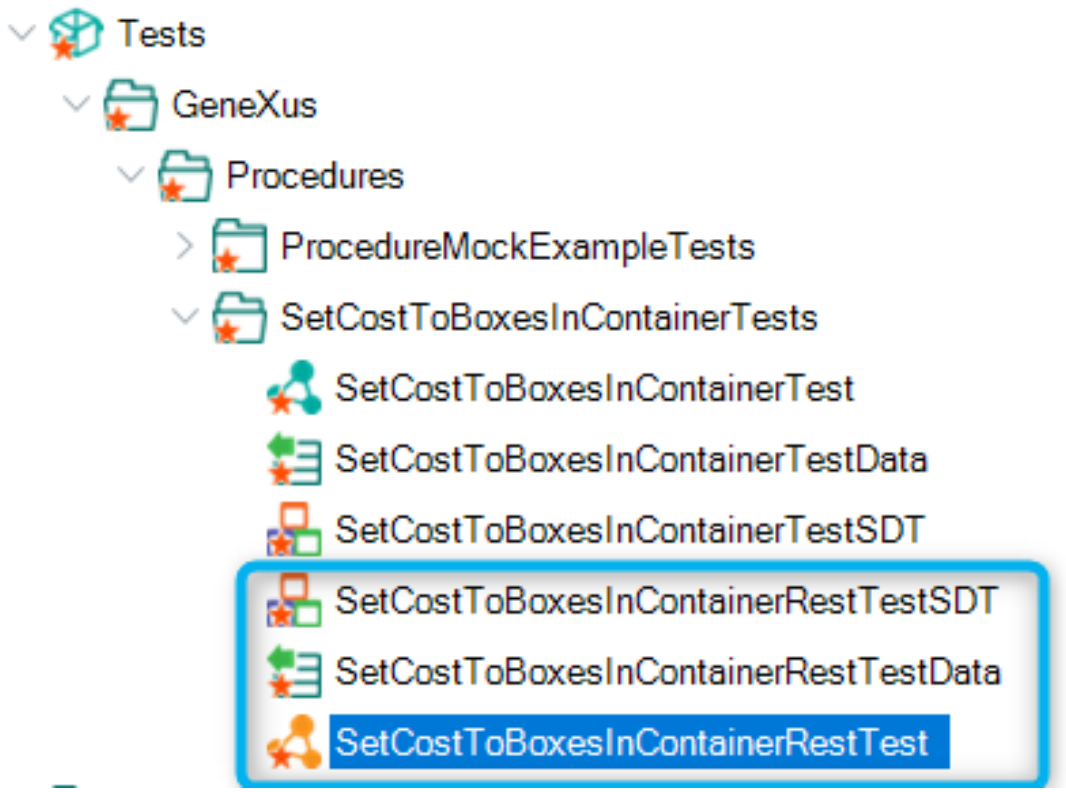
5) GAM のインポート完了後、KB エクスプローラ上で”SetCostToBoxesInContainer”オブジェクトを右クリックし、「Rest Test を作成」を選択します。



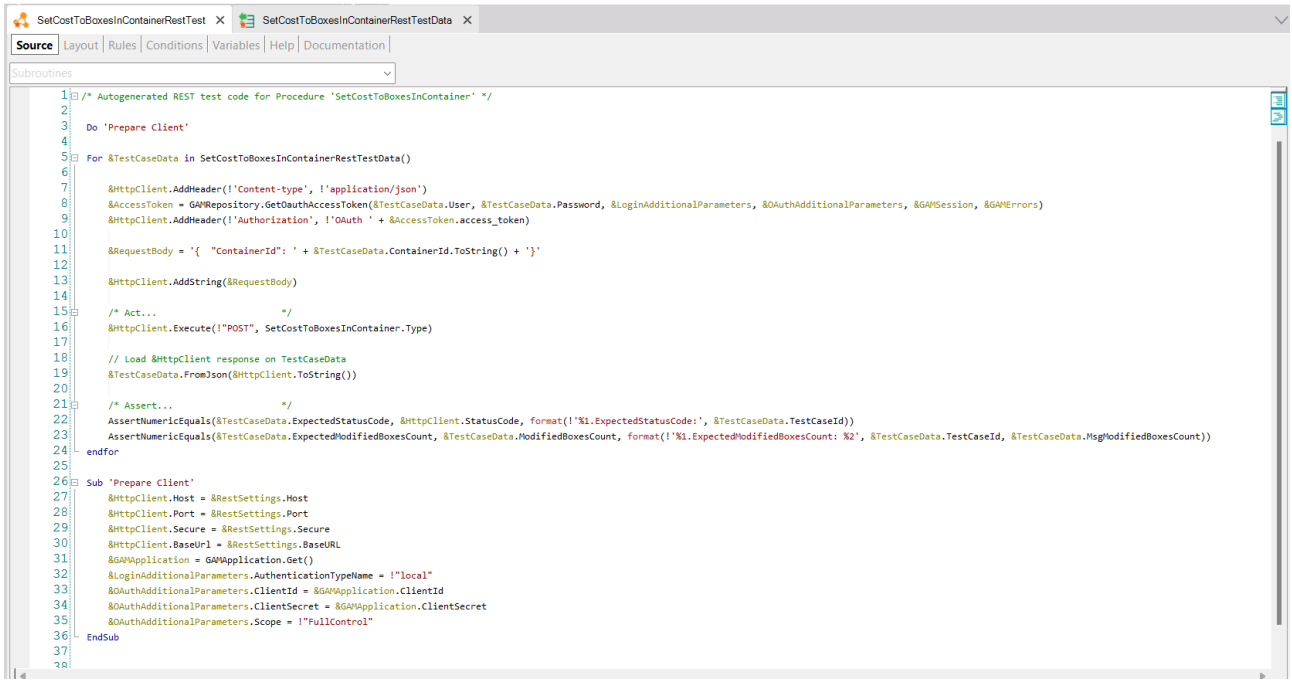
6) 以下のテストオブジェクトが作成されるまでお待ちください。

"SetCostToBoxesInContainerRestTest","SetCostToBoxesInContainerRestTestData", "SetCostToBoxesInContainerRestTestSDT"

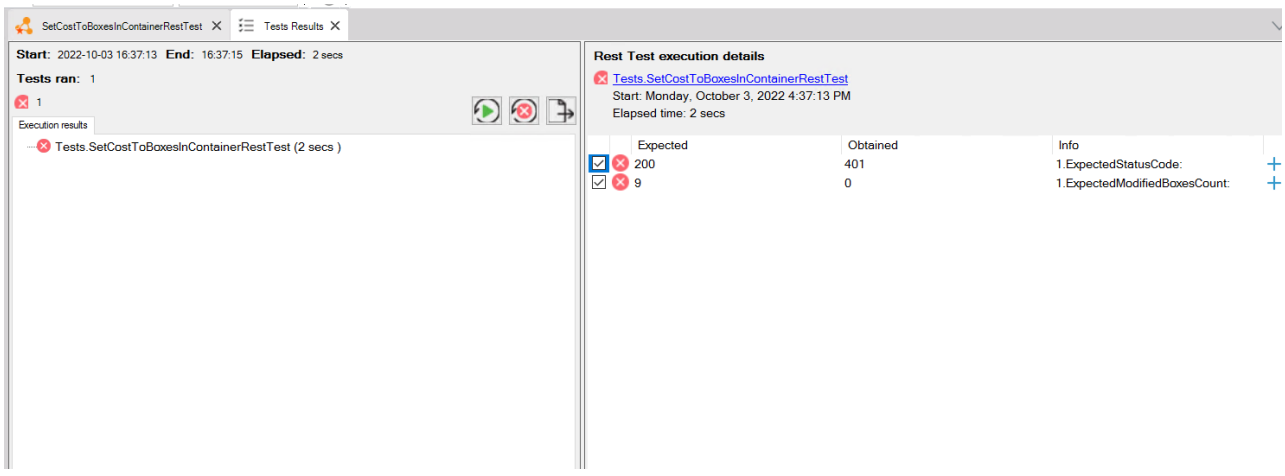
これらは、"Tests"モジュール内の"SetCostToBoxesInContainerTests"プロシージャーのフォルダの中に表示されます。



ご覧の通り、GXtest は自動的に"SetCostToBoxesInContainerRestTestData"データプロバイダの反復処理やテスト実行用のデータセットを含む GeneXus コードテンプレートを作成します。



7) 右クリックで「テストを実行」を選択し、"SetCostToBoxesInContainerRestTest"を実行します。



8) 実行されたデータセットのステータスコードを確認します。認証に必要なクレデンシャルを入力しなかったため 401 エラーコードが返り、サービスを利用することができませんでした。

GAM 認証

認証付きの Rest Test を実行するには、資格情報が有効でコード 200 が取得できるように、データプロバイダの値を変更する必要があります。

1) "SetCostToBoxesInContainerRestTestData"オブジェクト内で編集するフィールドを特定します。
2) GAM の認証情報を入力し、変更を保存して、「テストを実行」を選択してテストを実行します。

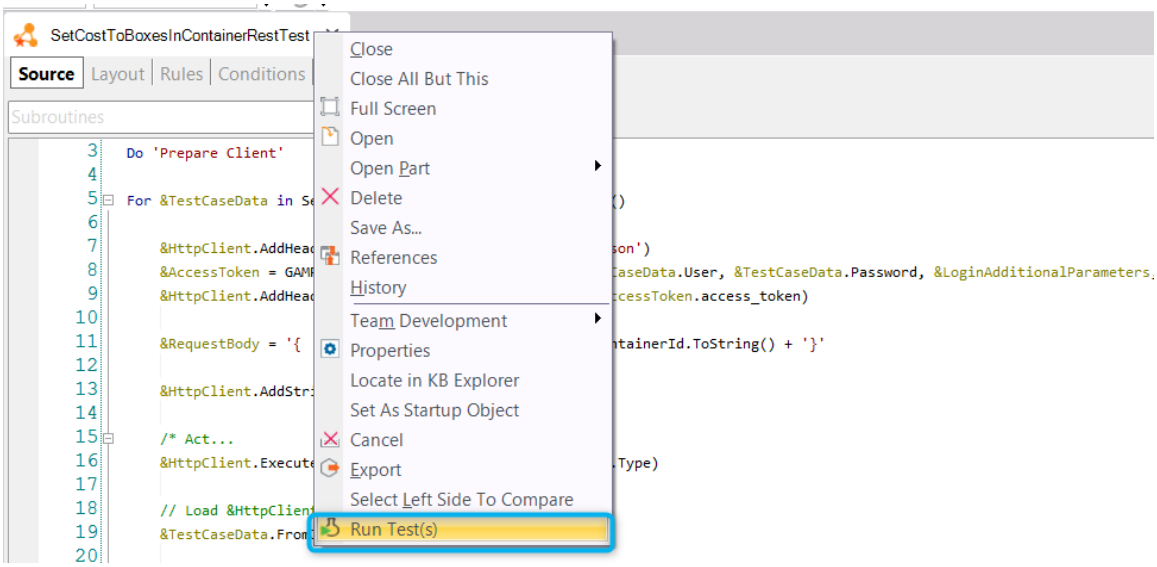
テストは成功し、アサーション結果と HTTP からの返値であるコード 200 が表示され、認証が成功したことが確認できます。

実行例:

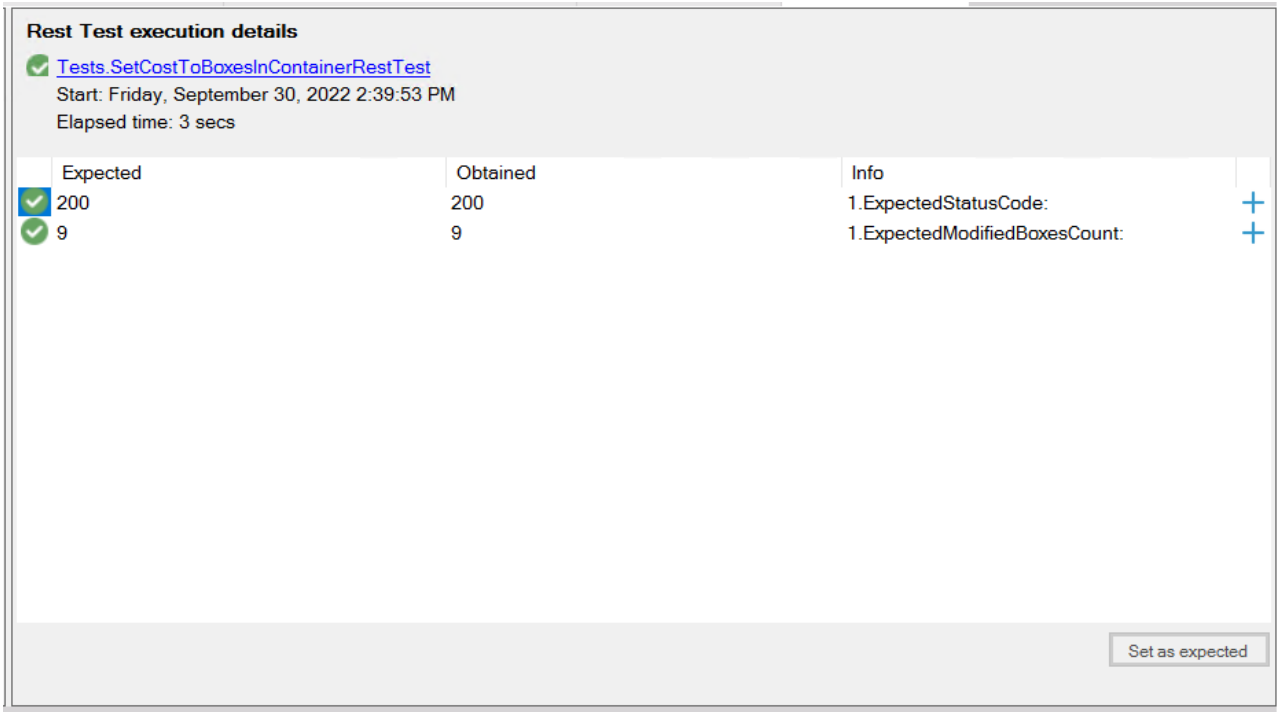
1) “SetCostToBoxesInContainerRestTestData”オブジェクトを以下のように変更します：

```
SetCostToBoxesInContainerRestTestSDT
{
        TestCaseId = '1'
        ExpectedStatusCode = 200
        User = 'admin'
        Password = 'admin123'
        ContainerId = 1
        ExpectedModifiedBoxesCount = 9
        MsgModifiedBoxesCount = '9'
}
```

2) “SetCostToBoxesInContainerRestTestData”オブジェクトの変更を保存し、Rest Test オブジェクトを右クリックし、「テストを実行」を選択します。



3) テストの実行が終了すると、「テスト結果」に結果が表示されます。認証が成功し、期待通りのステータスコードが得られ、修正ボックスの数も期待通りであったため、テストに合格したことが分かります。



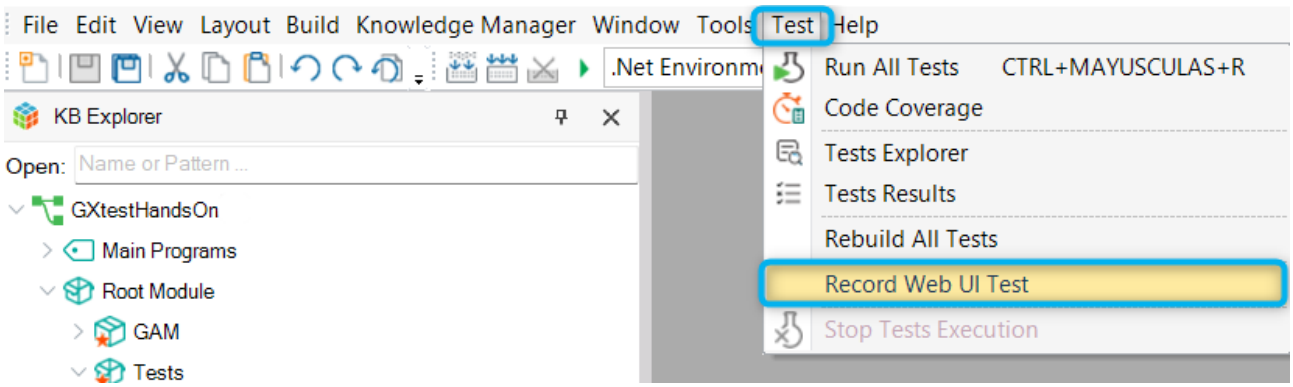
UI テスト

このセクションでは、これまで作業してきた Web アプリケーションで、ユーザー・インターフェース・テスト（以下、「UI テスト」）の実行についてご案内します。

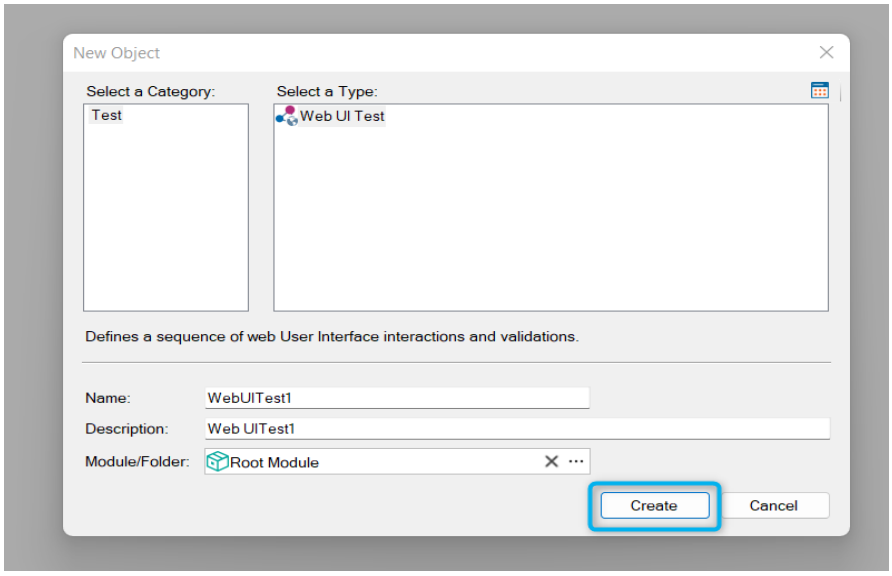
コンテナ 1 に属するボックスを作成し、コンテナ 1 のコストを設定し、コストが割り当てられたことを確認し、ボックスを削除する、という一連の流れを記録します。

GXtest Recorder によるシーケンス録画

1) メニューバーの「テスト」から、「Web UI テストを記録」を選択します。

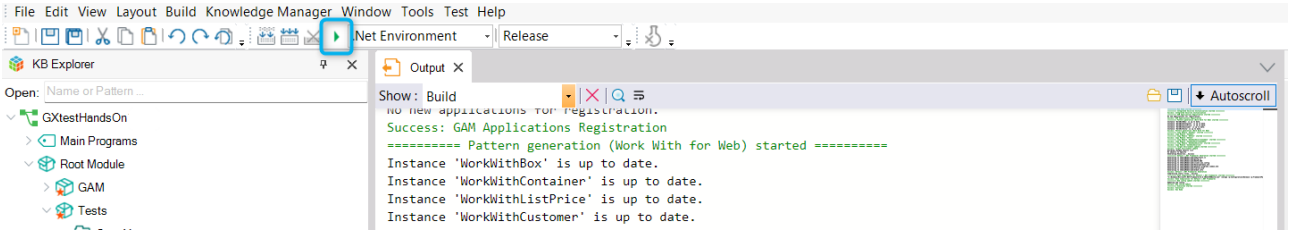


2) "Web UI Test"オブジェクトを新規作成するウィンドウが表示されます。
「作成」をクリックします。

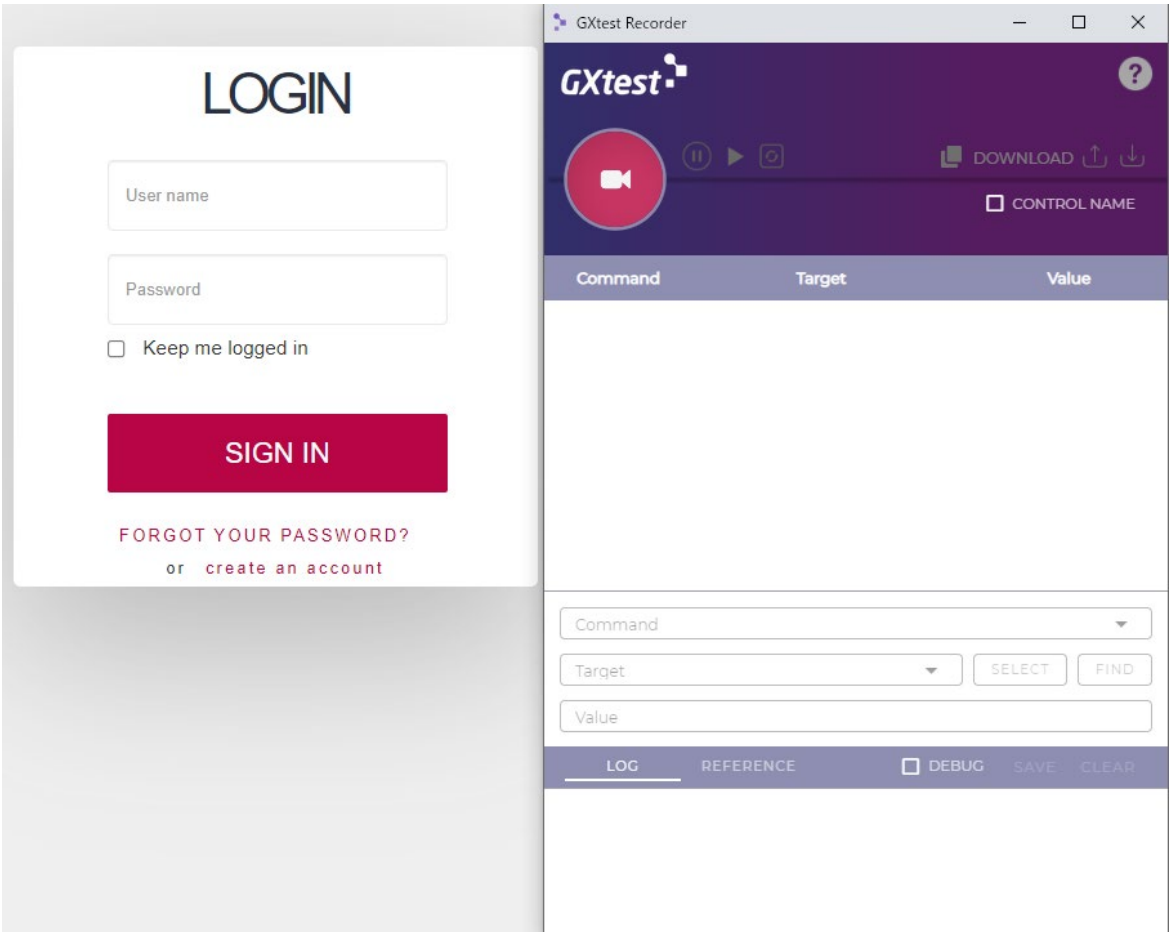


ブラウザが起動し、GXtest Recorder 拡張機能が録画開始の準備に入ります。

3) GeneXus IDE からアプリケーションを実行します（F5）。Chrome ブラウザが立ち上がり、アプリケーションが実行されるのを待ちます。



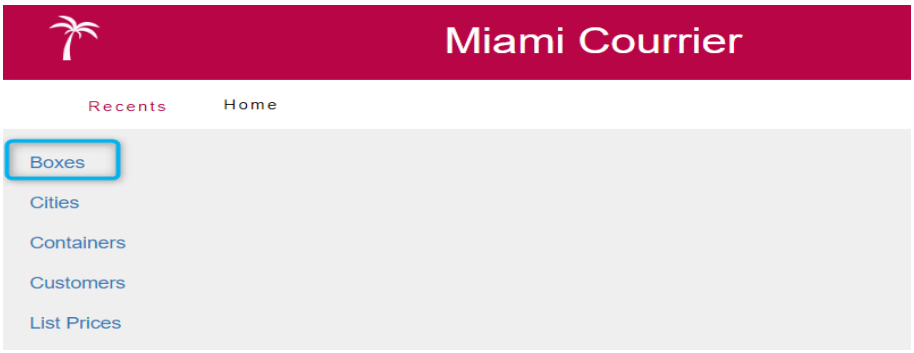
4) 録画を開始する前に、GXtest Recorder 拡張機能にて「CONTROL NAME」にチェックを入れておくと、レコーダーがこれらのセレクターを使用するようになります（使用できる場合のみ）。



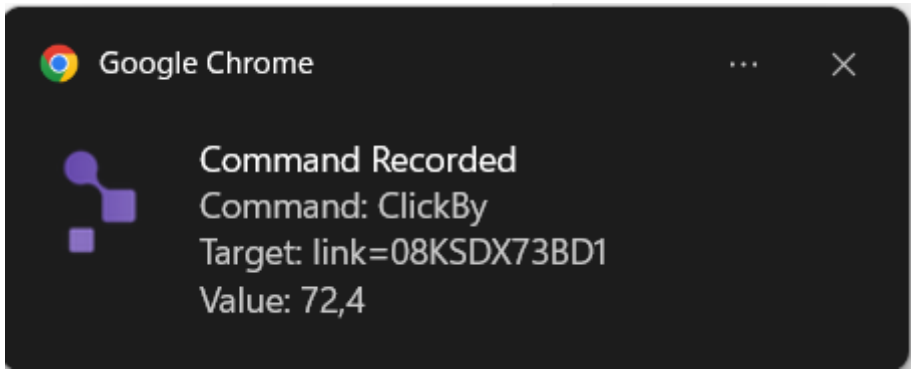
それでは、シーケンス録画を始めましょう。

1) User = **admin** および Password = **admin123** を入力し、「SIGN IN」ボタンをクリックします。アプリケーションのログインが記録されます。

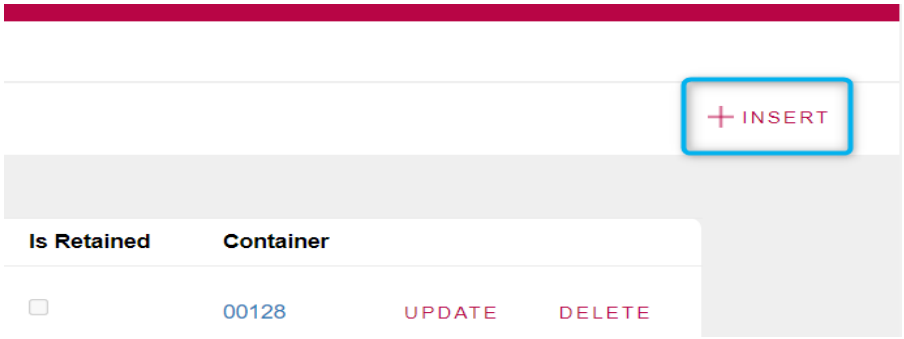
2) メインメニューの「Boxes」をクリックします。



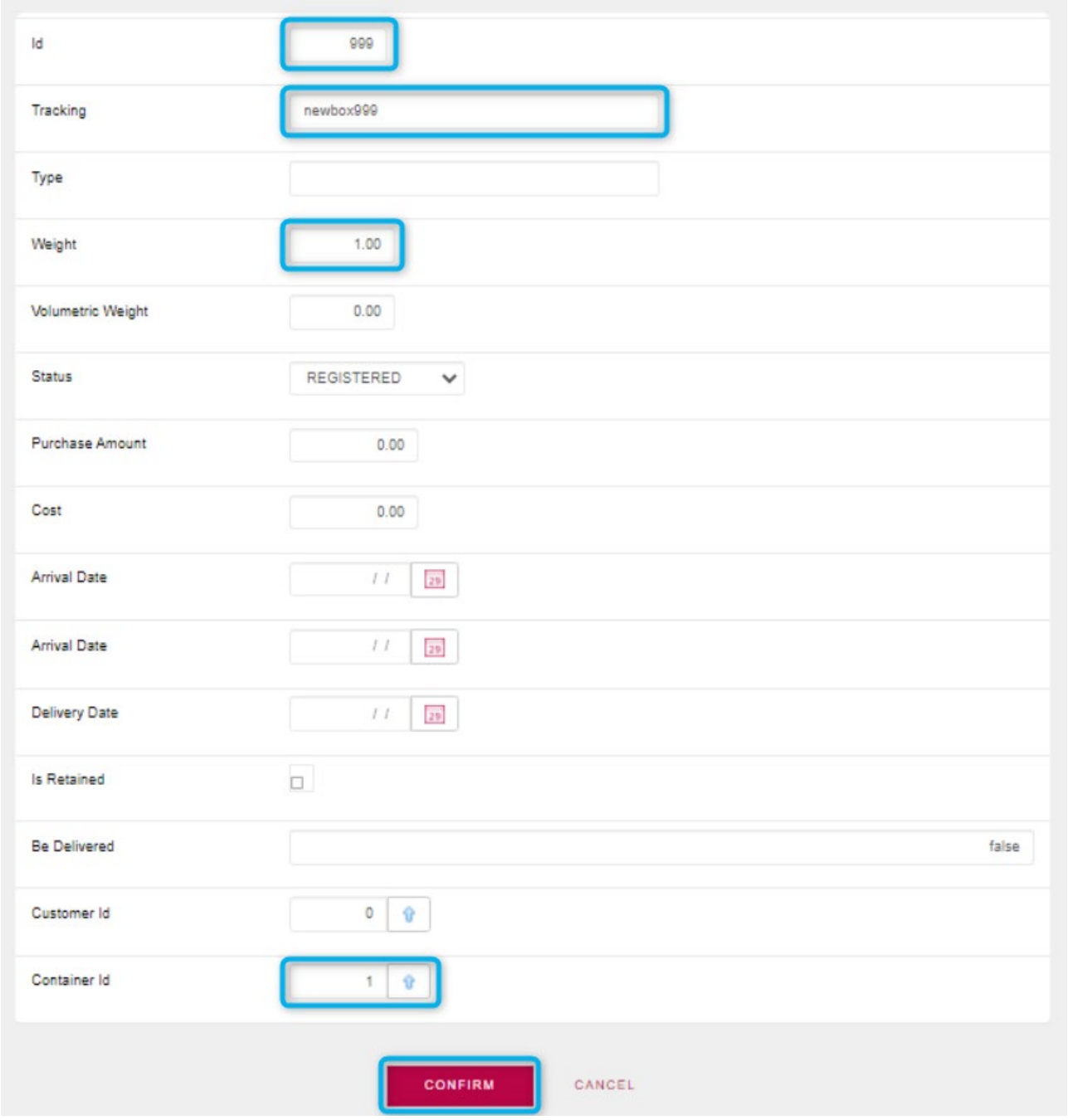
シーケンスが記録されていることは、画面右下に表示される通知を見ることで確認できます。記録された各アクションに対して、次のようなメッセージが表示されます。:



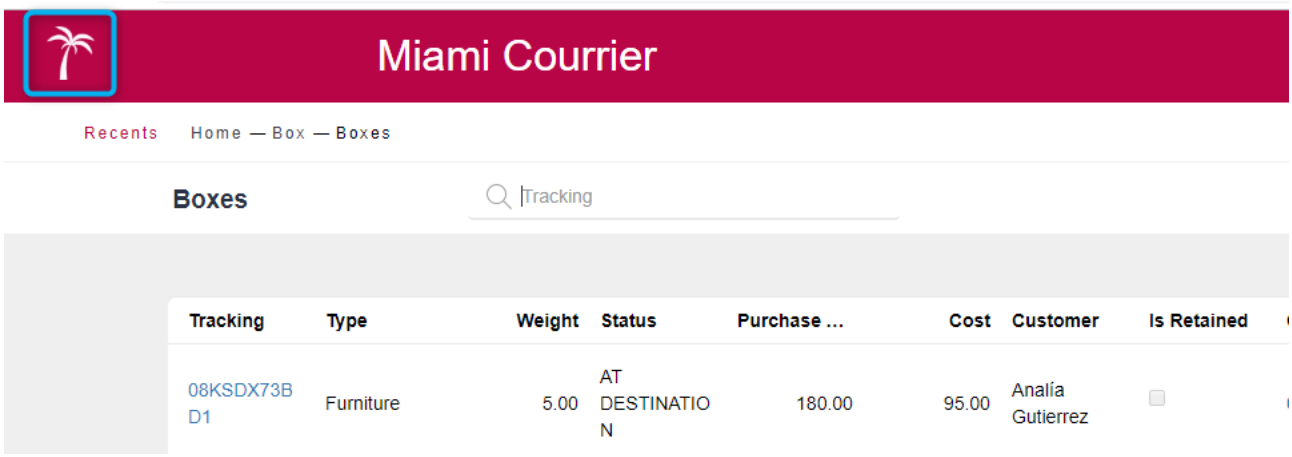
「INSERT」をクリックして、新しいボックスを作成します。



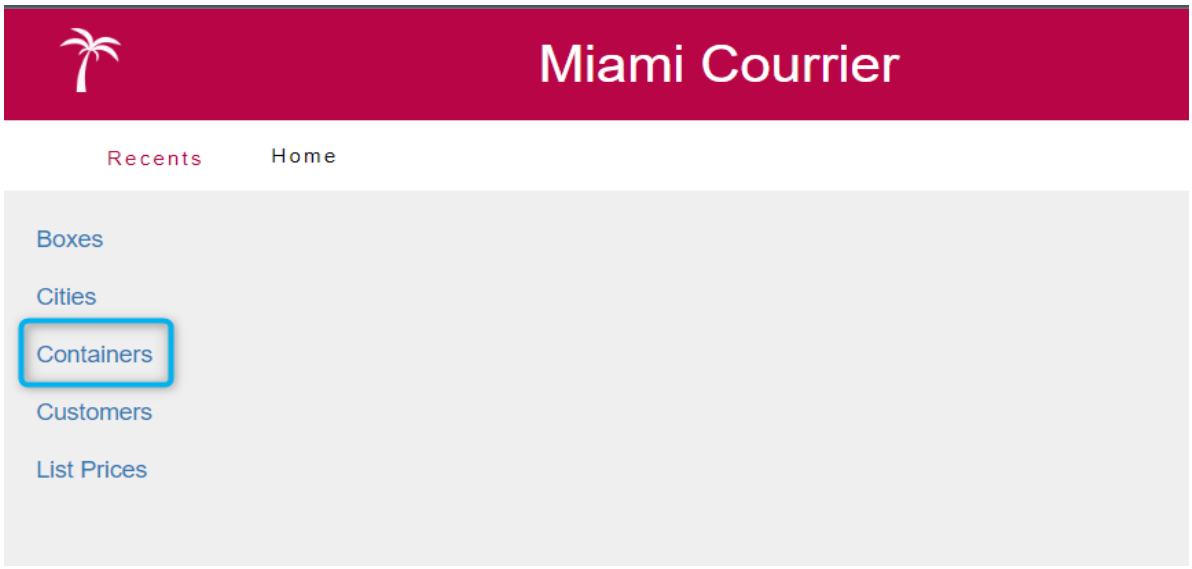
このボックスの Id=999, Tracking= NEWBOX999, Weight= 1, Container Id= 1 欄のみを記入し、「CONFIRM」をクリックします。



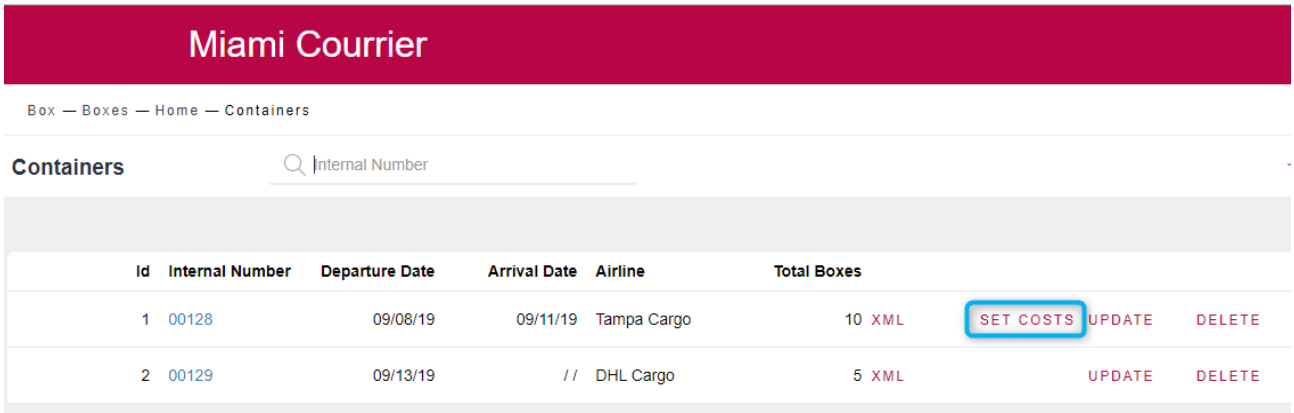
ヤシの木をクリックすると、トップページに戻ります。



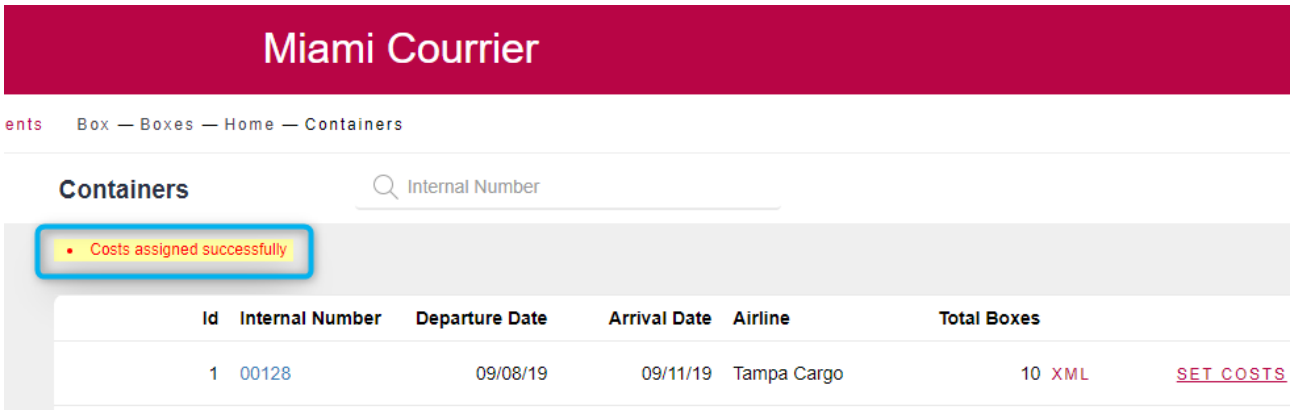
メニューから「Containers」を選択します。



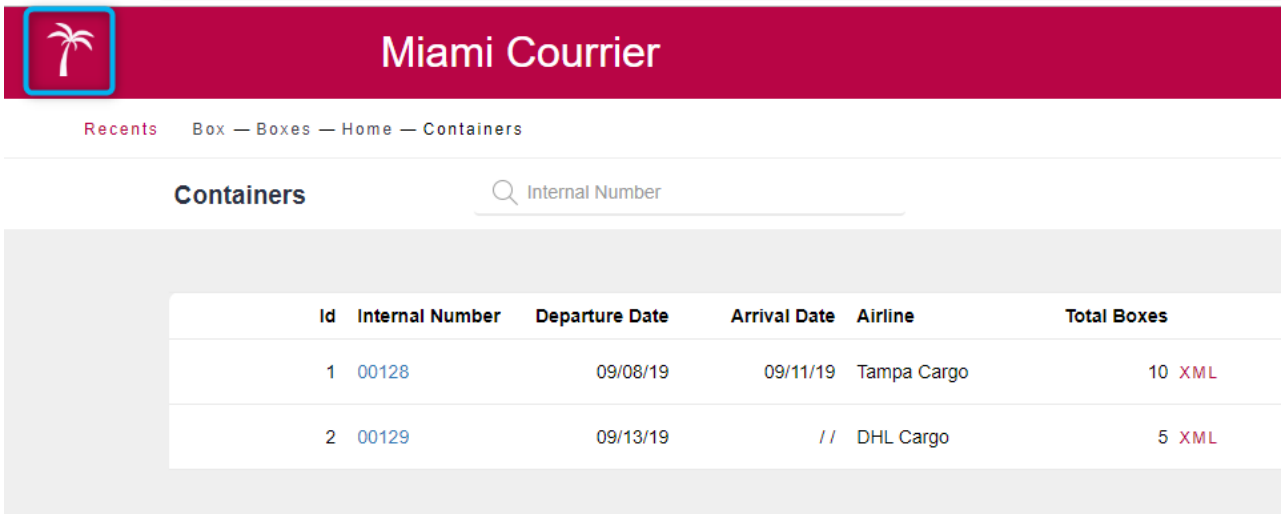
「SET COSTS」をクリックします。



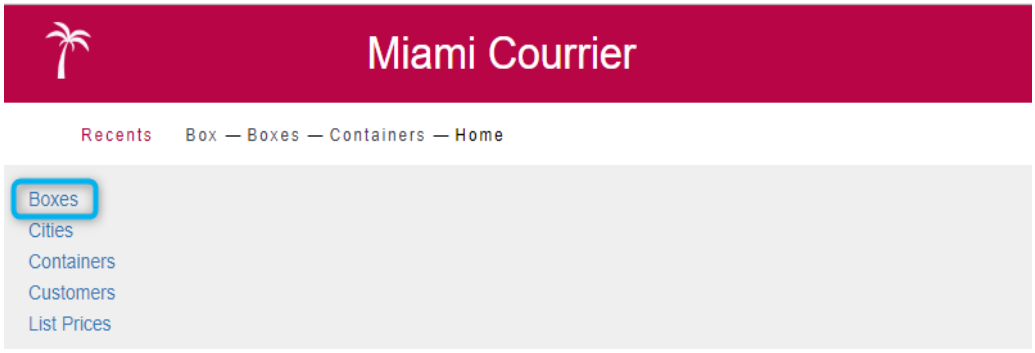
“Costs assigned successfully”（コストの割り当てに成功しました）と表示されます。



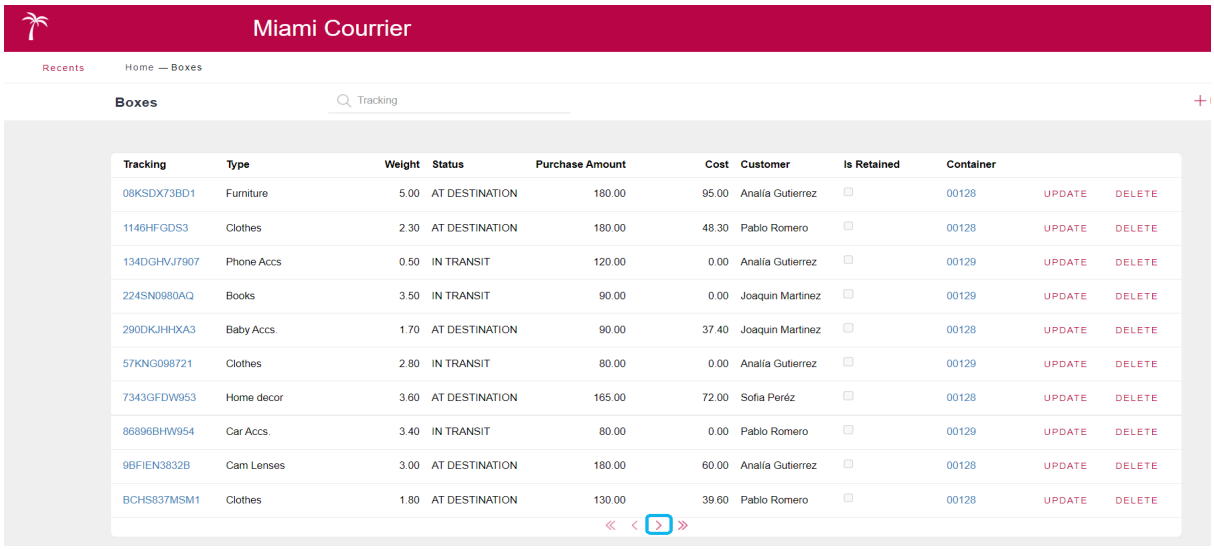
ヤシの木をクリックし、再びトップページに戻ります。



メインメニューの「Boxes」をクリックします。

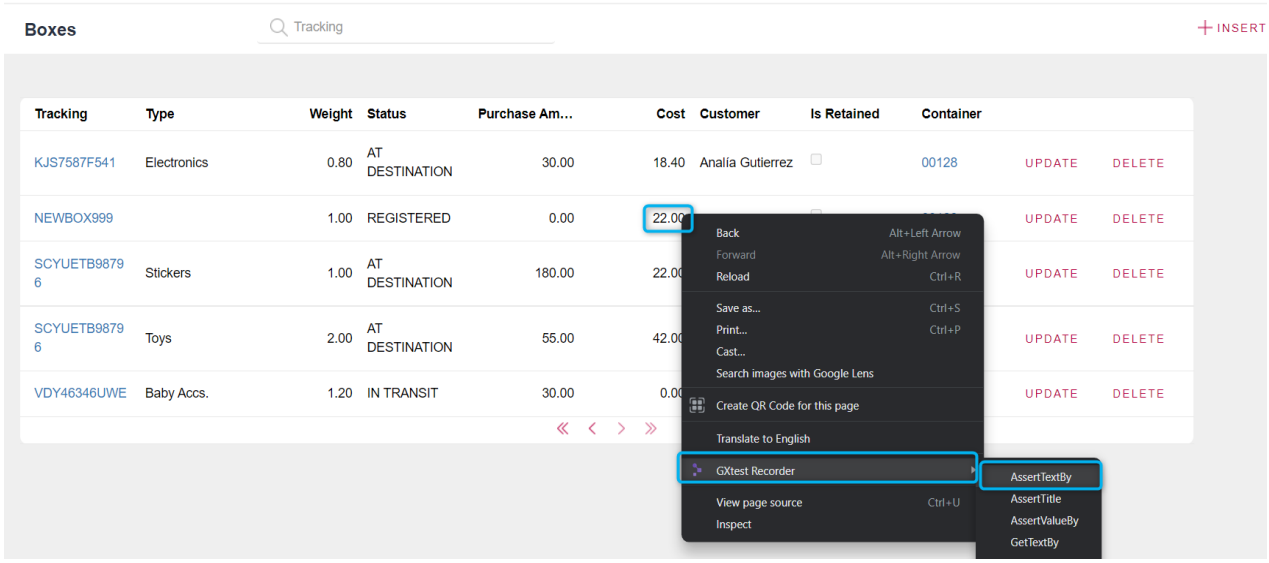


ページ下部の「>」をクリックし、次の画面へ遷移します。

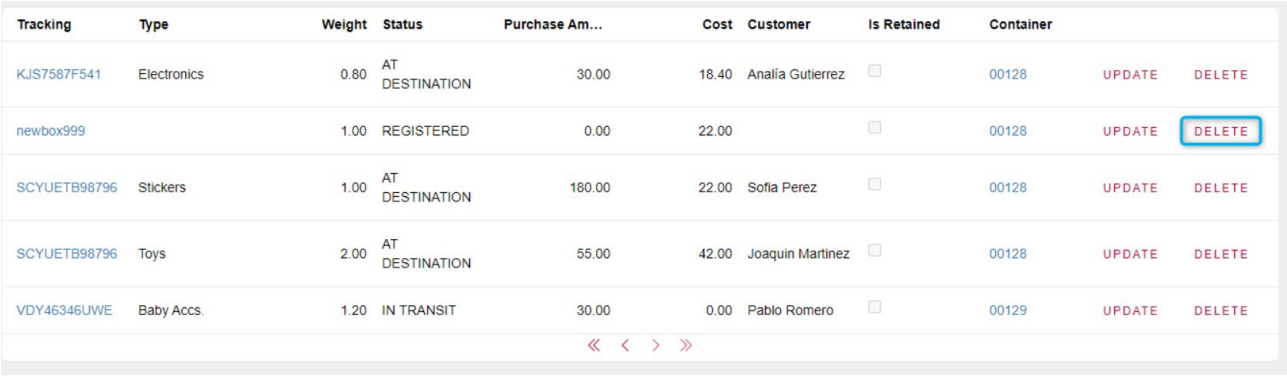


検証する値 22.00（この場合、”newbox999”ボックスの”Cost”の値）の上で右クリックします。

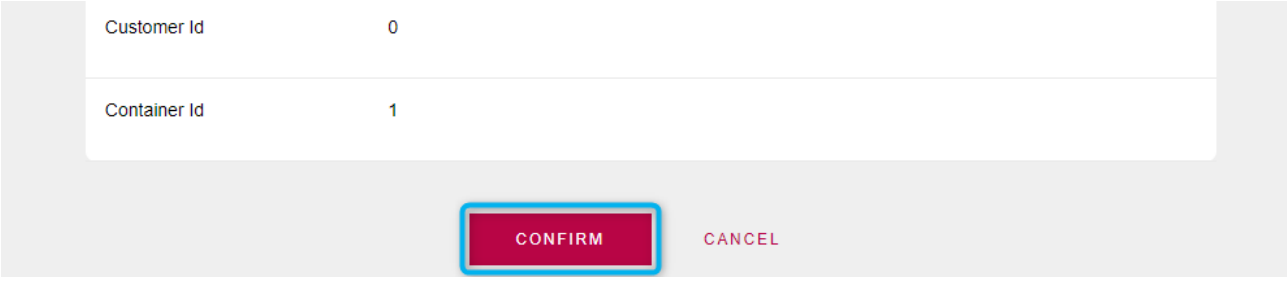
「GXtest Recorder」オプションの中で、「AssertTextBy」を選択します。



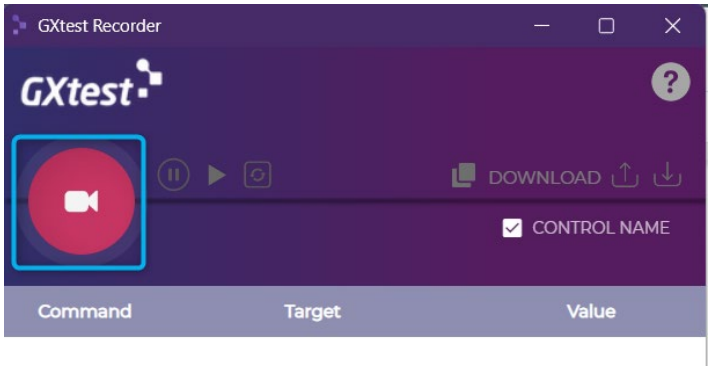
ボックス「newbox999」の「DELETE」をクリックし、「CONFIRM」をクリックします。



Tracking	Type	Weight	Status	Purchase Am...	Cost	Customer	Is Retained	Container		
KJS7587F541	Electronics	0.80	AT DESTINATION	30.00	18.40	Analía Gutierrez	☐	00128	UPDATE	DELETE
newbox999		1.00	REGISTERED	0.00	22.00		☐	00128	UPDATE	DELETE
SCYUETB98796	Stickers	1.00	AT DESTINATION	180.00	22.00	Sofia Perez	☐	00128	UPDATE	DELETE
SCYUETB98796	Toys	2.00	AT DESTINATION	55.00	42.00	Joaquin Martinez	☐	00128	UPDATE	DELETE
VDY46346UWE	Baby Accs.	1.20	IN TRANSIT	30.00	0.00	Pablo Romero	☐	00129	UPDATE	DELETE

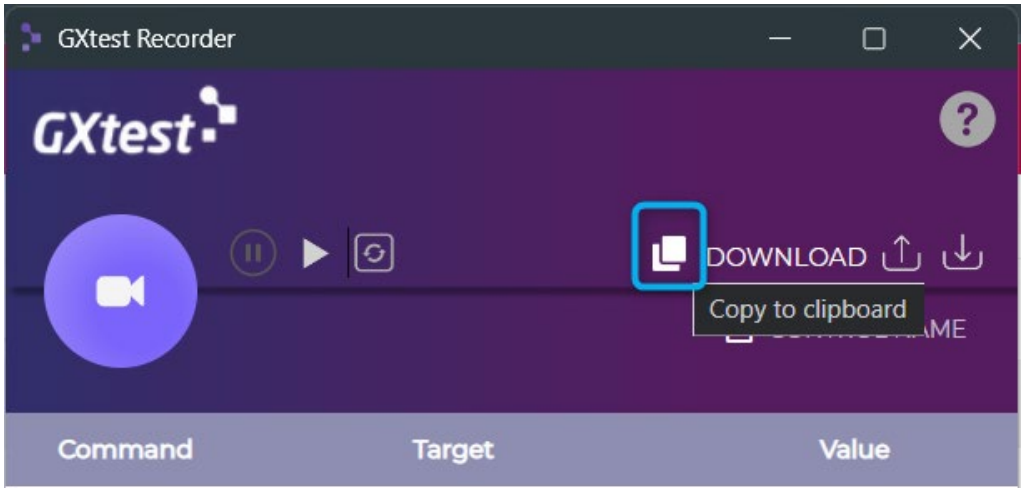


「Stop recording」ボタンを押して、録画を停止します。

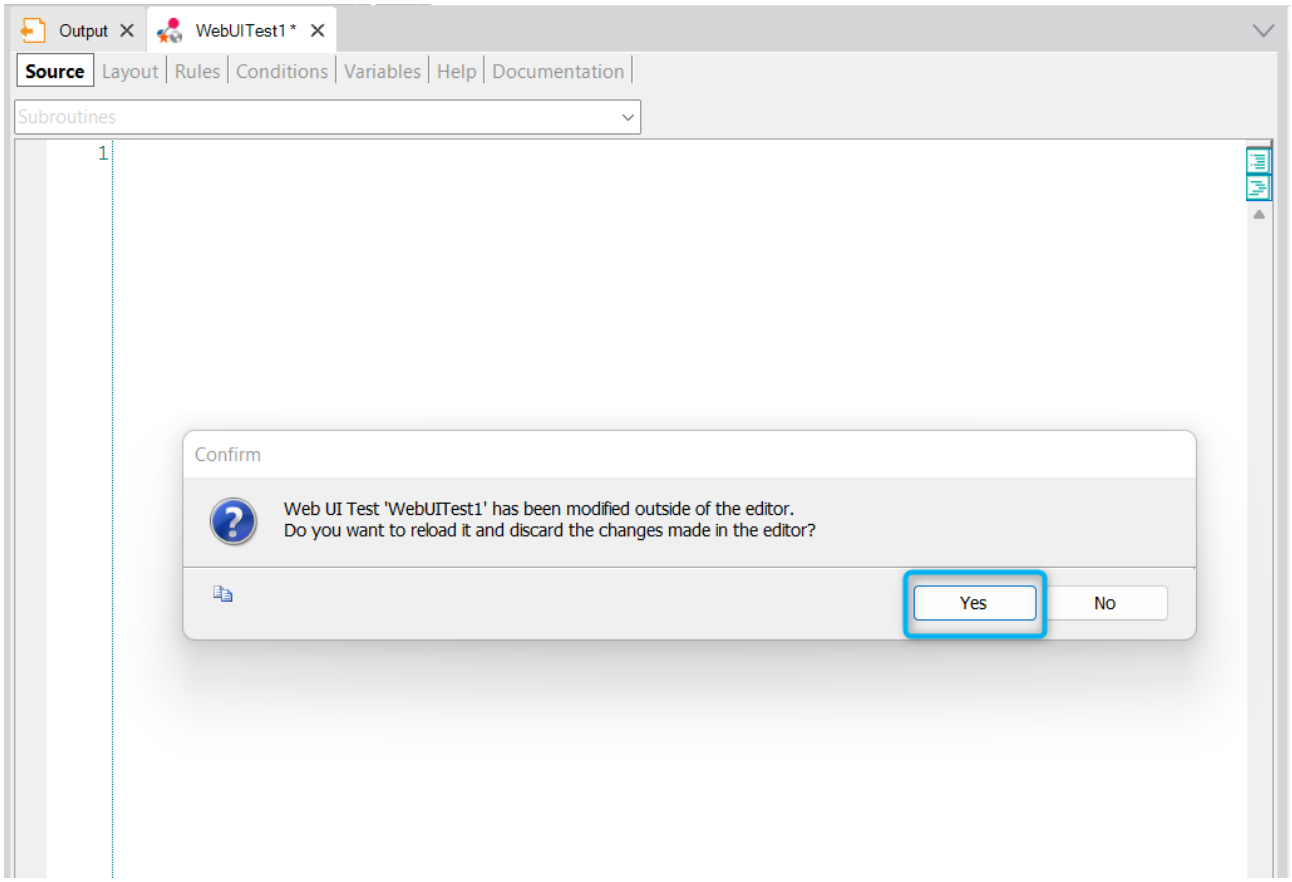


UI テストを KB にインポートする

1) 「Copy to clipboard」ボタンをクリックし、UI テストコードをエクスポートします。



2) IDE に戻ります。IDE 外でテストが変更された旨のメッセージが表示され、オブジェクトを読み込むかどうか確認が求められます。「はい」をクリックして承諾します。



3) GXtest Recorder 拡張機能でキャプチャしたスクリプトを実行可能な状態とするため、Go コマンドの URL を更新し、テストオブジェクトを更新します。Go コマンドは、ホームパネルの URL(home.aspx)をパラメータとして受け取る必要があります。

Source | Layout | Rules | Conditions | Variables | Help | Documentation

Subroutines

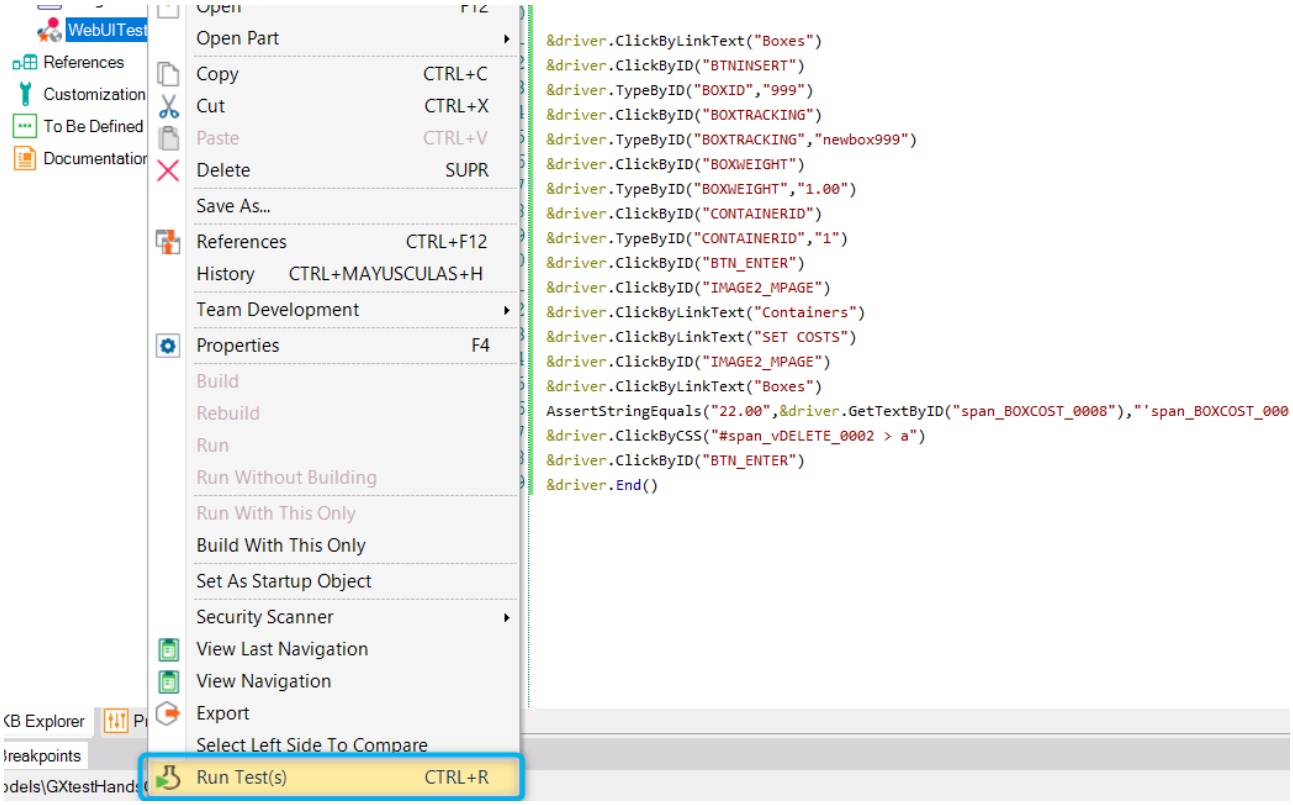
```
// Script generated using GXtest Recorder

//Start webdriver
&driver.Start()
&driver.Maximize()

// Initial navigation
&driver.Go("http://localhost/GXtestHandsOnTraining.NetEnvironment/home.aspx")

&driver.Click("&username")
&driver.Type("&username","admin")
&driver.Type("&userpassword","admin123")
&driver.Click("login")
&driver.ClickByLinkText("Boxes")
&driver.Click("btninsert")
&driver.Type("boxtracking","NEWBOX999")
&driver.Click("boxweight")
&driver.Type("boxweight","1.00")
&driver.Click("containerid")
&driver.Type("containerid","1")
&driver.Click("btn_enter")
&driver.ClickByID("IMAGE2_MPAGE")
&driver.ClickByLinkText("Containers")
&driver.ClickByLinkText("SET COSTS")
&driver.ClickByID("IMAGE2_MPAGE")
&driver.ClickByLinkText("Boxes")
&driver.ClickByCSS("button[class='PagingButtonsNext btn btn-default']")
AssertStringEquals("22.00",&driver.GetText("boxcost",2),"'boxcost',2 not matching 22.00")
&driver.ClickByCSS("#span_vDELETE_0002 > a")
&driver.Click("btn_enter")
&driver.End()

```

生成されるコードの例は以下の通りです:

```
// Script generated using GXtest Recorder

//Start webdriver
&driver.Start()
&driver.Maximize()

// Initial navigation
//update the URL before running the test!!!!
&driver.Go("http://localhost/GXtestHandsOnTraining.NetEnvironment/home.aspx")

&driver.Click("&username")
&driver.Type("&username","admin")
&driver.Type("&userpassword","admin123")
&driver.Click("login")
&driver.ClickByLinkText("Boxes")
&driver.Click("btninsert")
&driver.Type("boxtracking","NEWBOX999")
&driver.Click("boxweight")
&driver.Type("boxweight","1.00")
&driver.Click("containerid")
&driver.Type("containerid","1")
&driver.Click("btn_enter")
&driver.ClickByID("IMAGE2_MPAGE")
&driver.ClickByLinkText("Containers")
&driver.ClickByLinkText("SET COSTS")
&driver.ClickByID("IMAGE2_MPAGE")
&driver.ClickByLinkText("Boxes")
&driver.ClickByCSS("button[class='PagingButtonsNext btn btn-default']")
AssertStringEquals("22.00",&driver.GetText("boxcost",2),"'boxcost',2 not matching 22.00")
```

```
&driver.ClickByCSS("#span_vDELETE_0002 > a")
&driver.Click("btn_enter")
&driver.End()
```

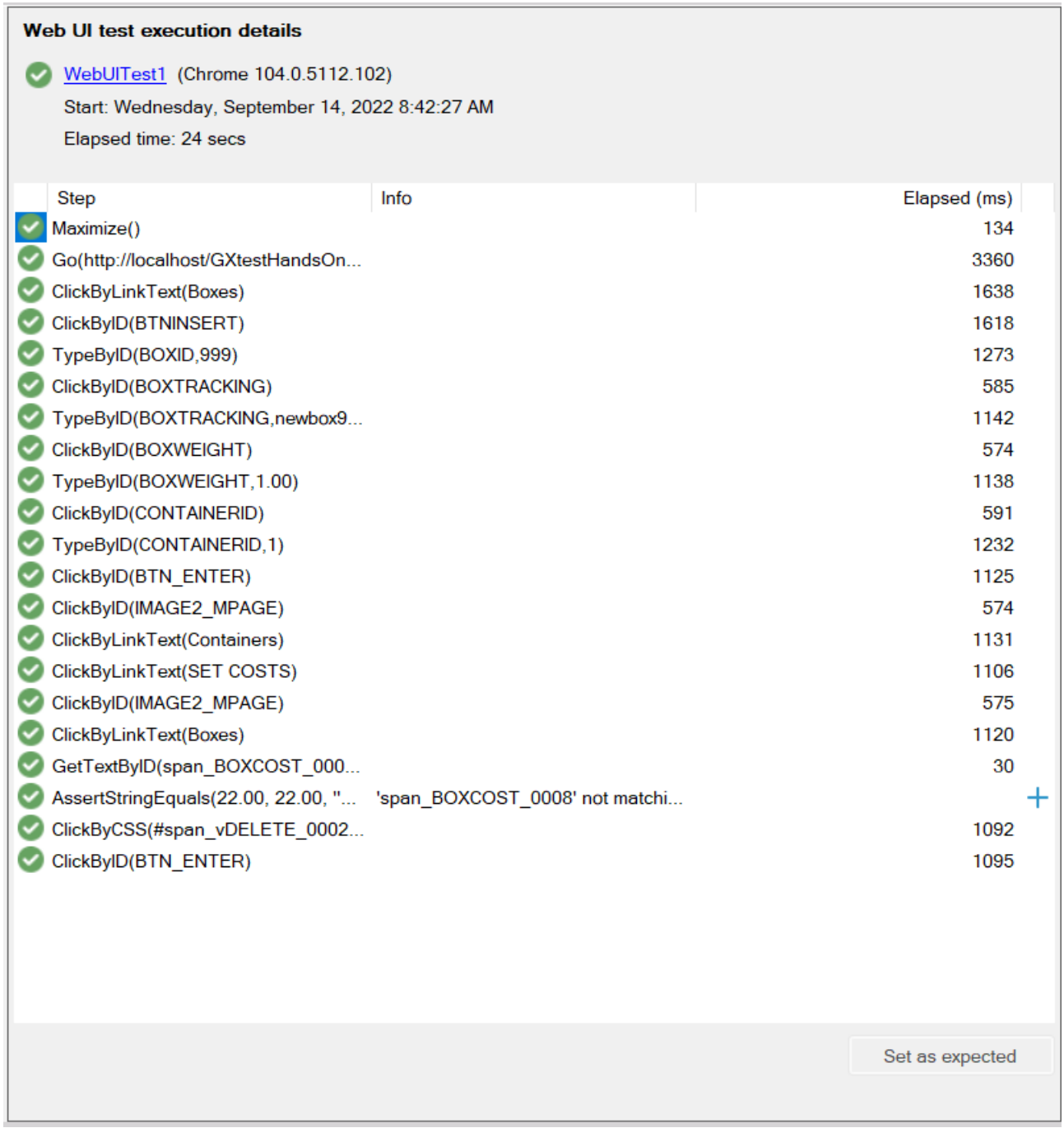
4）変更を保存します。オブジェクトを右クリックし、「テストを実行」を選択します。



実行に失敗した場合は、手順 3 で作成したコードと相違がないことを確認してください。

実行が終了したら、テストの結果を確認してください。

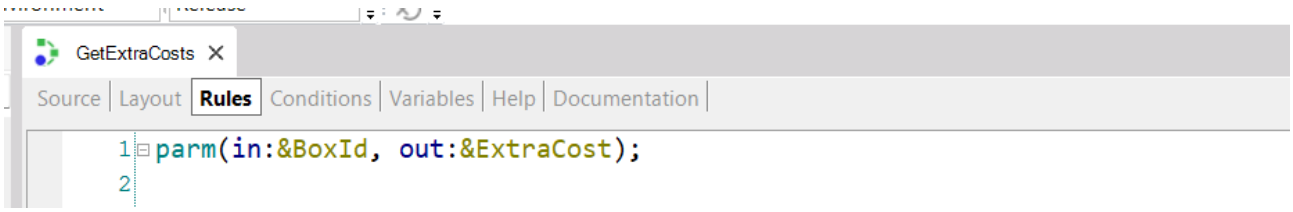
以下の「テスト結果」画面の内容を確認しましょう。



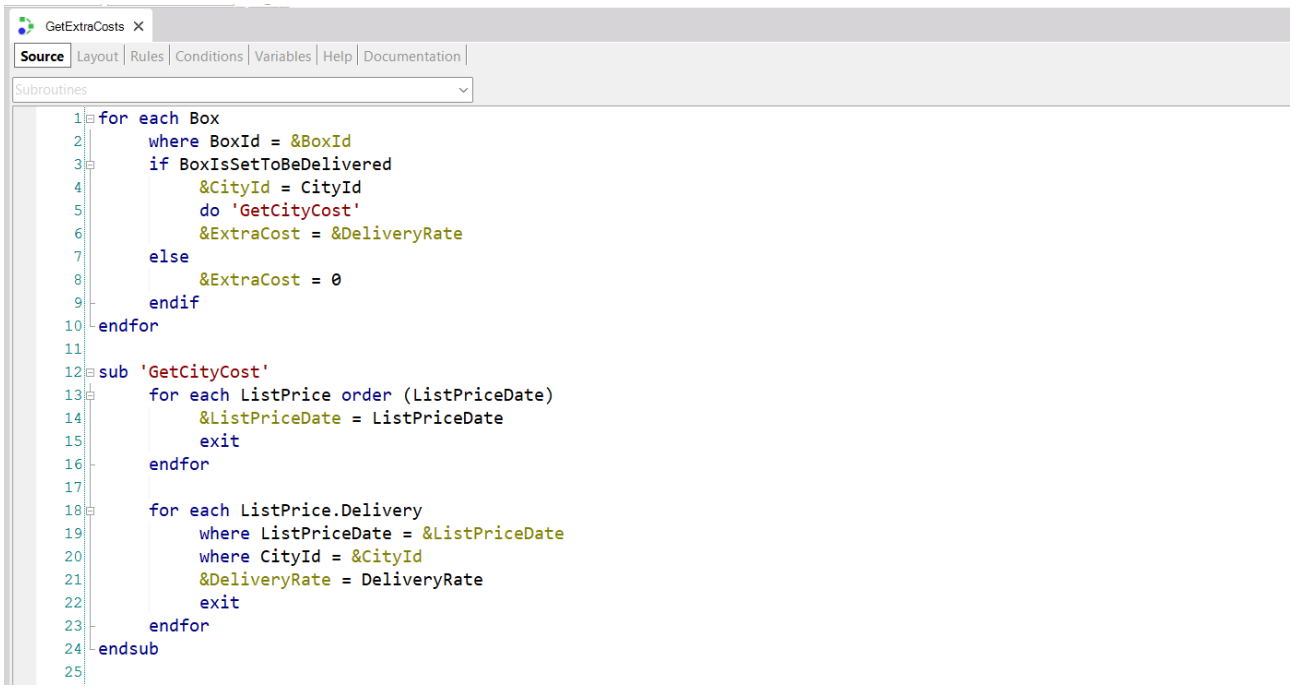
テストカバレッジ

テストカバレッジ機能を用いることで、Unit Test で実行される各オブジェクトの実行コード行数の割合を知ることができます。このセクションでは、KB 内の他のオブジェクトを呼び出す Unit Test を作成、実行し、そのカバレッジを分析する方法を説明します。

1) "GetExtraCost"プロシージャーを開き、”Rules”タブをクリックします。入力パラメータとして”&BoxId”（これは追加コストを取得するボックスの ID）を受け取り、出力パラメータとして”&ExtraCost”（出荷先に割り当てられた都市に応じた各ボックスの追加コスト）を返します。



2) "Source"タブをクリックします。このプロシージャーのコードは、入力パラメータ”&BoxId”と一致する各ボックスに対して、そのボックスが配送可能な状態である場合にのみ、追加コストを割り当てます。そうでない場合は、"ExtraCost = '0'"を返します。

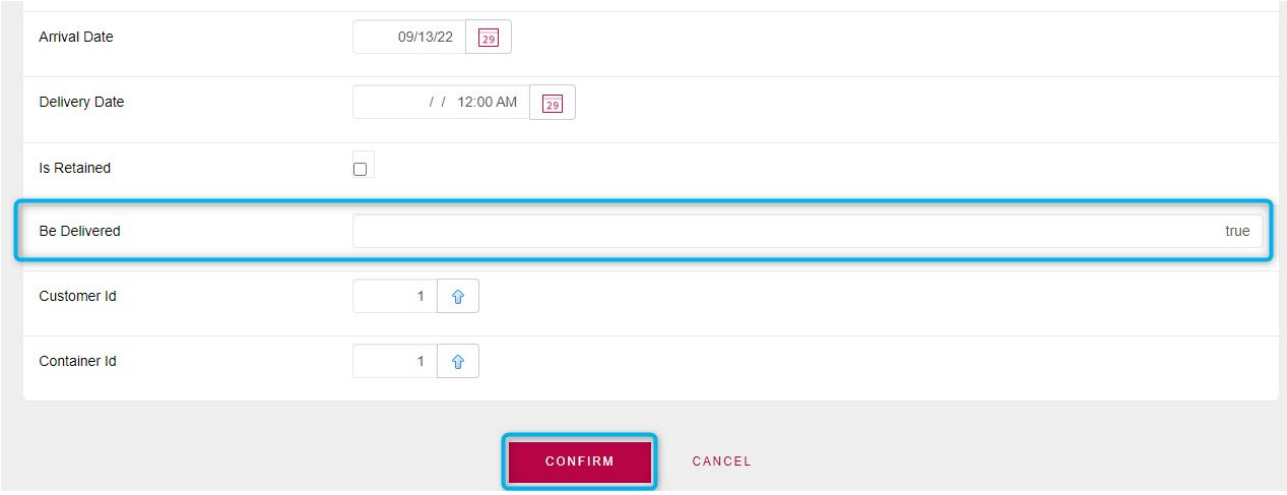


このプロシージャーには 24 行のコードが存在しますが、実際に評価され実行されるコードはこのうち 12 行です。

入力された”&BoxId”に対し、”BoxIsSetToBeDelivered”が”True”として代入されている場合は、この 12 行のうち 11 行のみが実行されます。そうでない場合は、else の中の 1 行だけが実行されます。

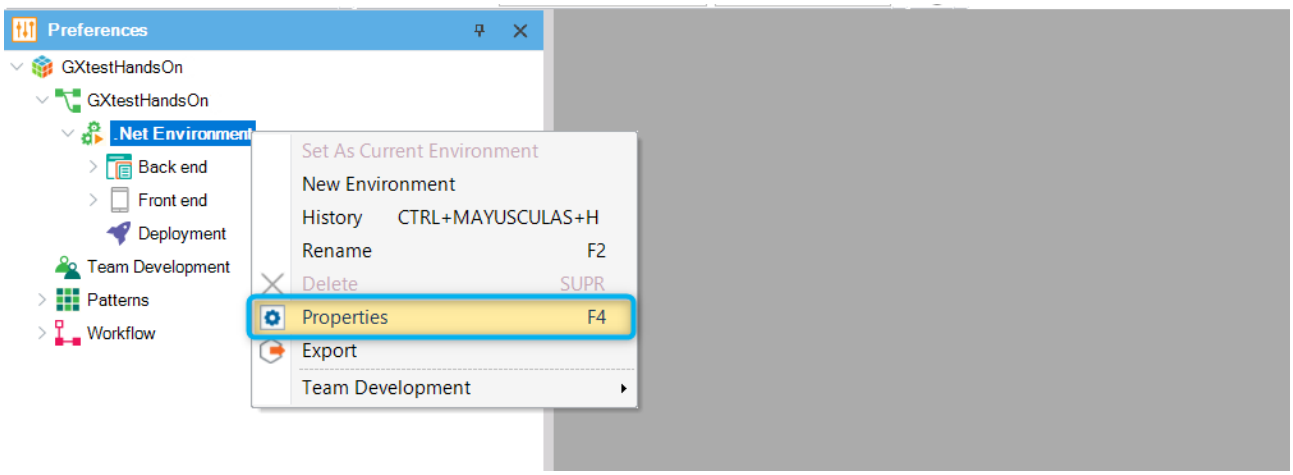
プロシージャーのコード全体をカバーするテストケースを作成し、トータル・カバレッジを目指します。

1) はじめにアプリケーションを開き、Id=1 の Box を開きます。"Be Delivered"セルが既定で”False”となっているため、"True"と書き込んで「CONFIRM」をクリックします。

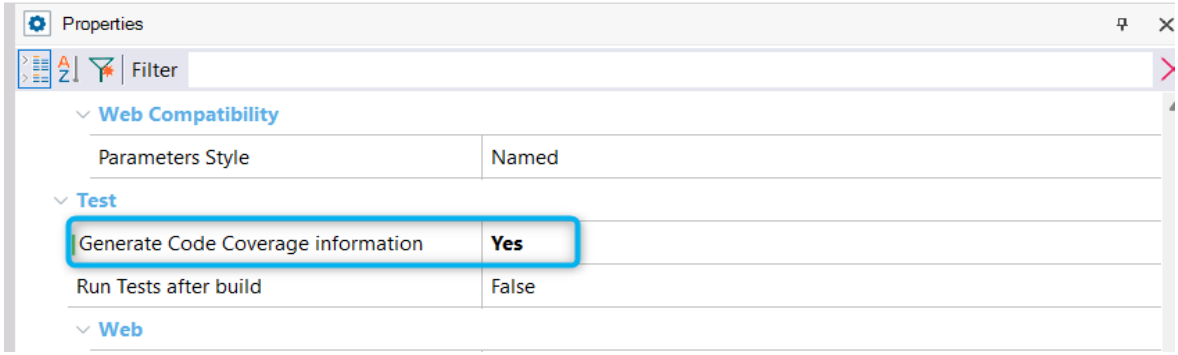


2) KB に戻り、”GetExtraCost”プロシージャーを右クリックし、「ユニットテストの作成」を選択して、ユニットテストを作成します。

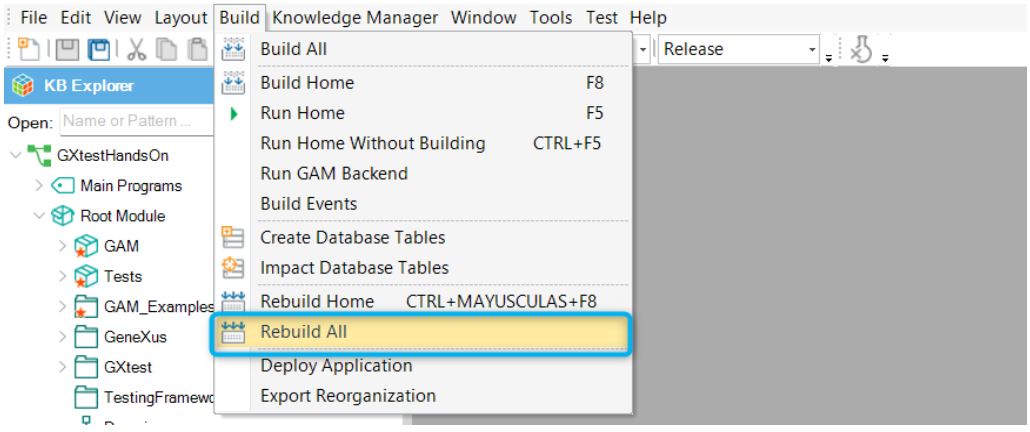
3) 「設定」にて三階層目を右クリックし「プロパティ」を選択し、環境のプロパティを開きます。



“Generate Code Coverage Information”プロパティを”Yes”に変更します。



4) 有効になったら、メニューバーの「ビルド」から「すべてリビルド」を選択します。このステップは、コードカバレッジトラッカーでオブジェクトを再作成するために必要です。

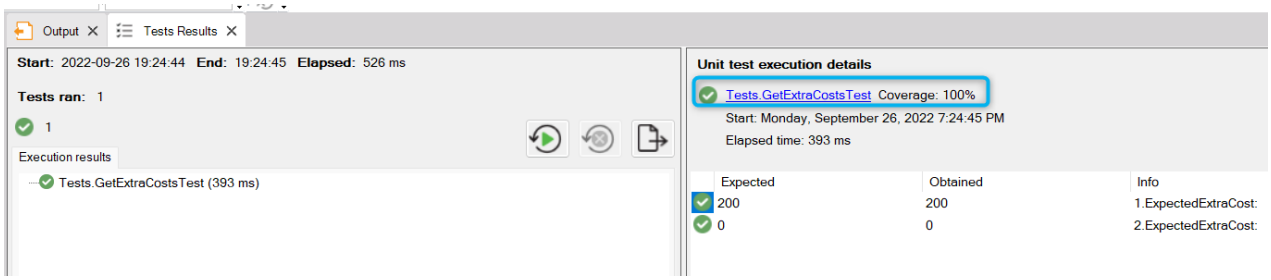


5）GetExtraCostsTestData を開きコードを以下のように変更します：

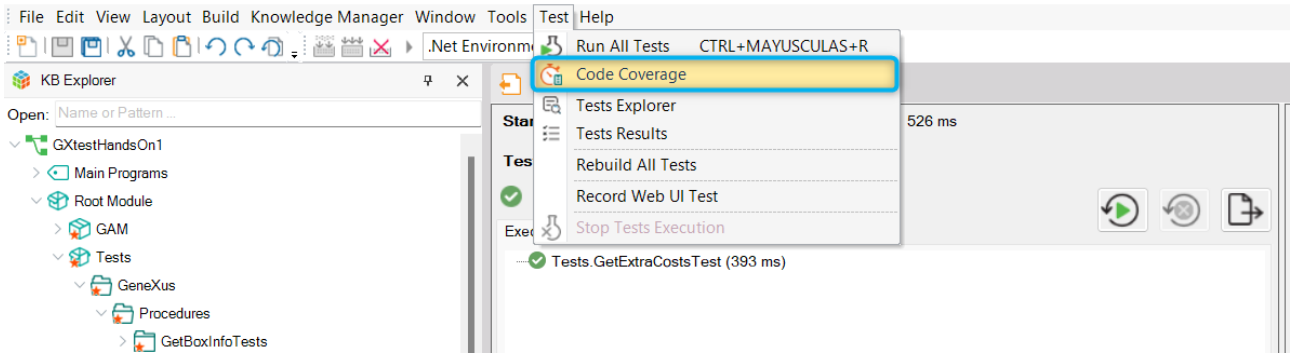
```
GetExtraCostsTestSDT
{
        TestCaseId = '1'
        BoxId = 1
        ExpectedExtraCost = 200
        MsgExtraCost = '200'
}
GetExtraCostsTestSDT
{
        TestCaseId = '2'
        BoxId = 2
        ExpectedExtraCost = 0
        MsgExtraCost = '0'
}
```

6）“GetExtraCostsTestData”の変更を保存して、"GetExtraCostsTest"を実行します。

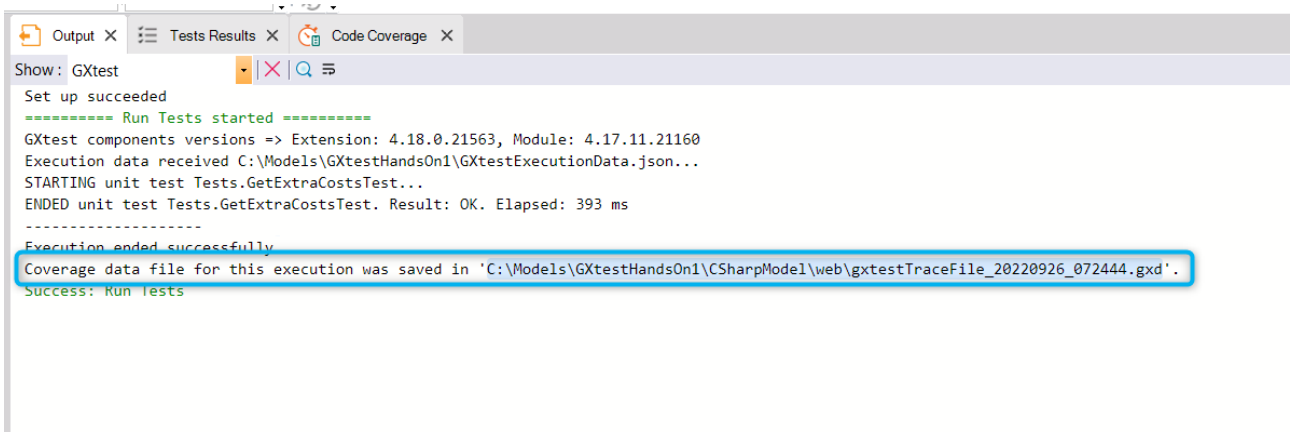
テストの実行が終了すると、その結果がテスト結果画面に表示されます。テストで実行されるオブジェクトのコードが 100%カバーされていることがわかります。



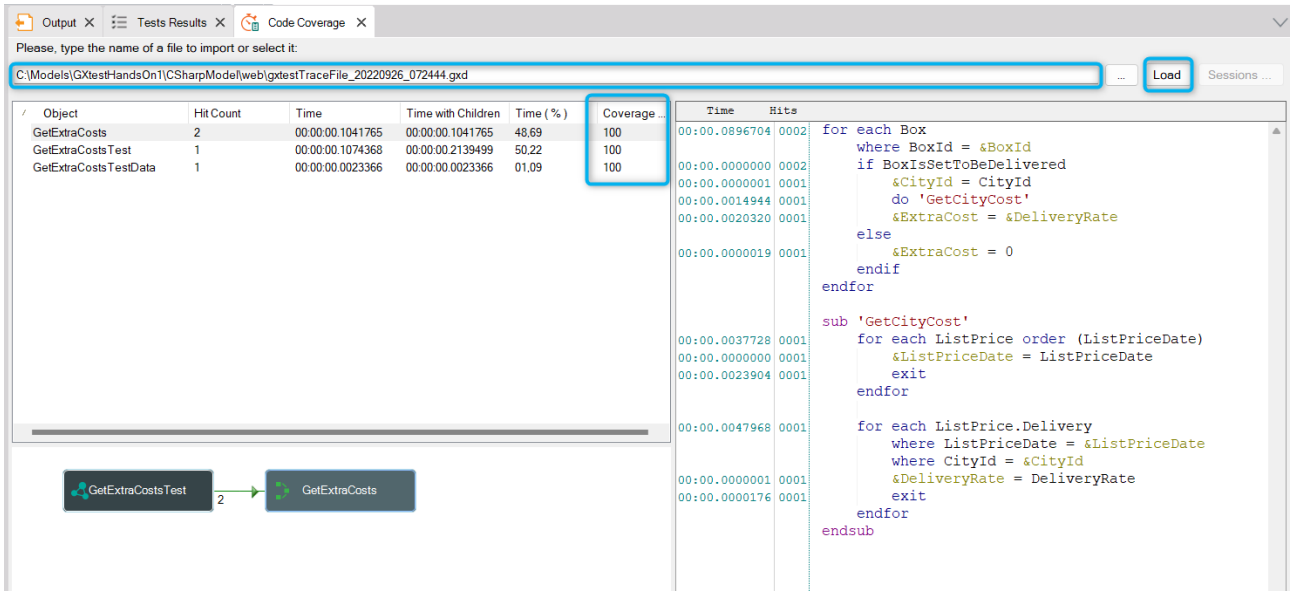
7) メニューバーの「テスト」から、「コードカバレッジ」をクリックします。



8) 「出力」にて、「表示」コンボボックスを「GXtest」に変更し、カバレッジデータが保存されたパスをコピーします。



9) "性能テスト"の中のフィールドにパスを貼り付け、「Load」をクリックします。



“Coverage”欄にカバレッジの割合が表示されます。

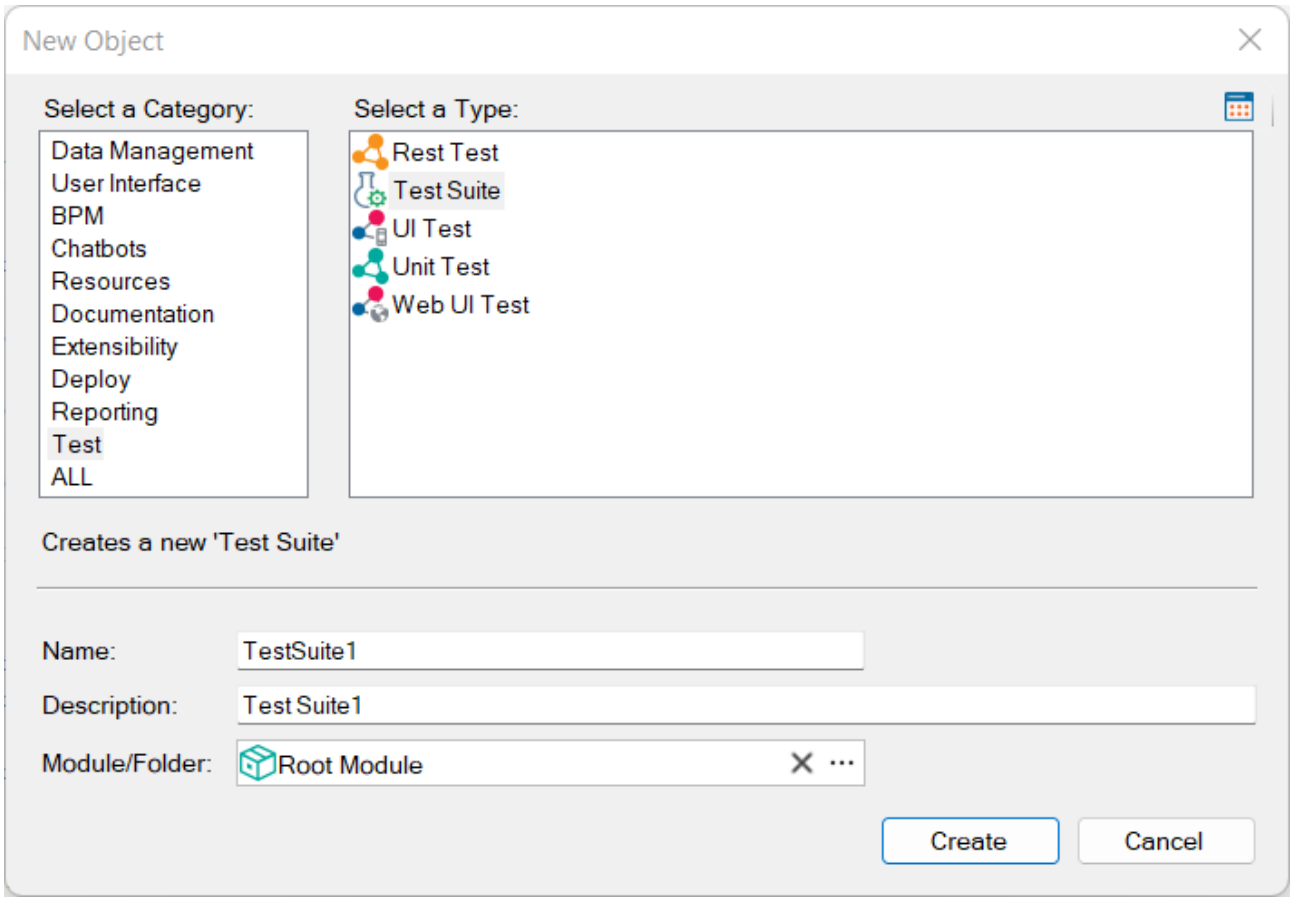
テストケースでは、最初の条件を満たす BoxId=1 が実行され、BoxId=2 が else 分の実行に相当するため、そのシナリオの 12 行をトレースし、BoxId=2 が else 文の実行に相当するため、コードの 100%がカバーされます。

テストスイート

テストスイートオブジェクトを作成することで、テストをグループ化し、特定の順序でユニットとして実行することが可能です。以下では、作成したテストを配置する Test Suite オブジェクトを実装します。

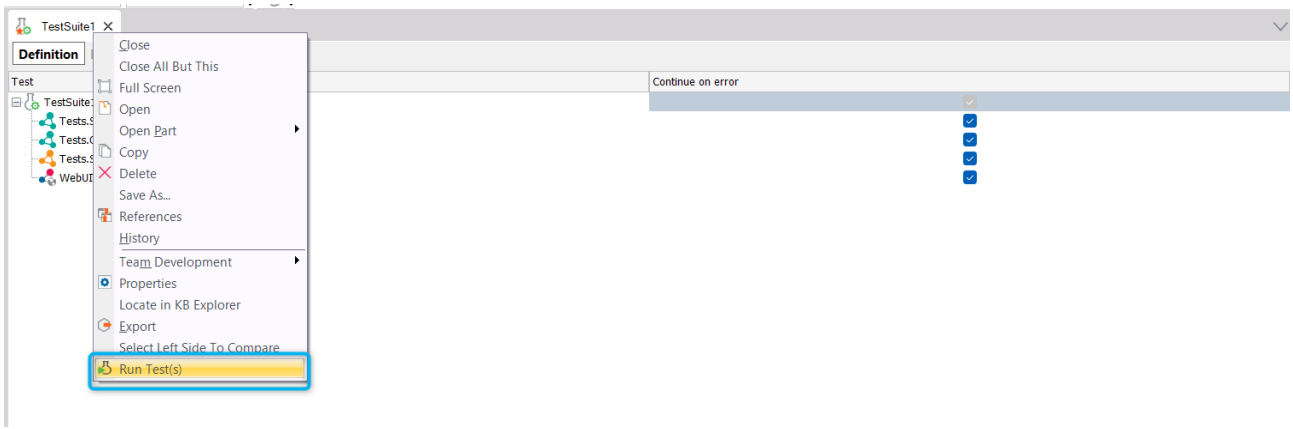
テストスイートの作成と実行

1)「ファイル」→「新規」→「オブジェクト」で、「テスト」カテゴリを選択し、”Test Suite”オブジェクトを選択し、「作成」を押して、テストスイートオブジェクトを作成します。



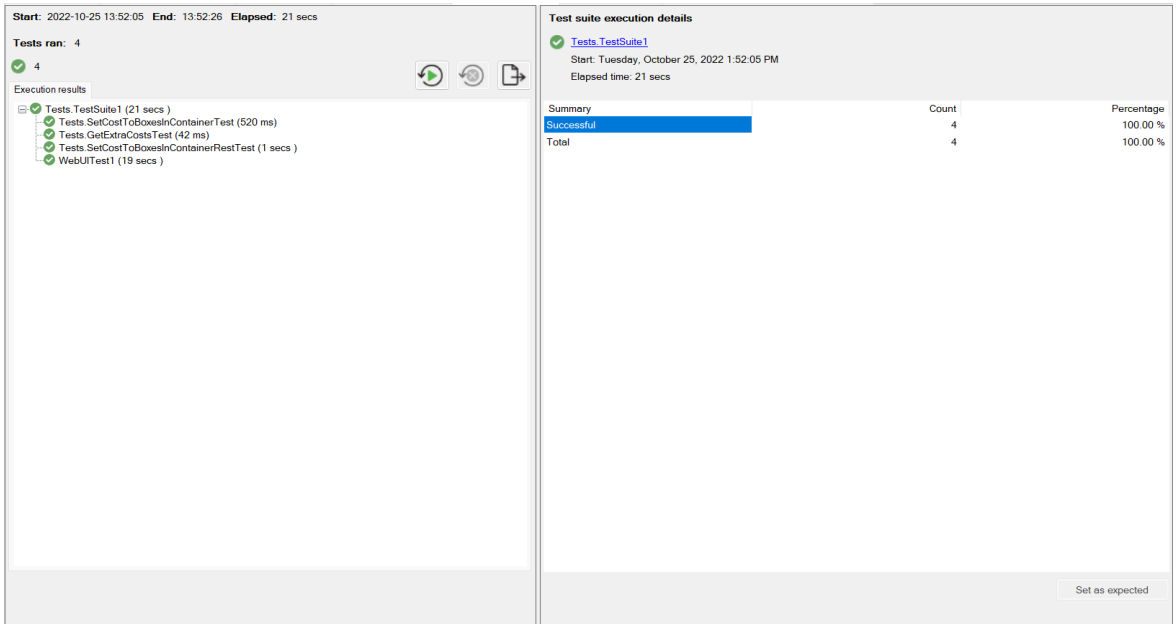
3) KB エクスプローラから、先に作成したユニットテストである”SetCostToBoxesInContainerTest”と”GetExtraCostTest”、Rest Test オブジェクトの”SetCostToBoxesInContainerRestTest”、Web UI テストの”WebUITest1”を選択してドラッグしてください。

3)作成した Test Suite オブジェクトを右クリックし、"テストを実行"をクリックします。



スイートに含まれるテストは、指定された順序で実行されます。

この場合、すべてのテストスイートのテストが合格しているので、実行は成功です。



GeneXus™ by Globant

MONTEVIDEO - URUGUAY	Av. Italia 6201- Edif. Los Pinos, P1	(598) 2601 2082
CIUDAD DE MÉXICO - MÉXICO	Hegel N° 221, Piso 2, Polanco V Secc.	(52) 55 5255 4733
MIAMI - USA	8950 SW 74th Ct., Suite 1406	(1) 201-603-2022
SÃO PAULO - BRASIL	Rua Samuel Morse 120 Conj. 141	(55) 11 4858 0300
TOKYO - JAPAN	2-27-3, Nishi-Gotanda	(81) 3 6303 9381
	Shinagawa-ku, Tokyo, 141-0031	(81) 3 6303 9980