Rest API Samples

Last updated: May 30, 2014

Contents

Introduction

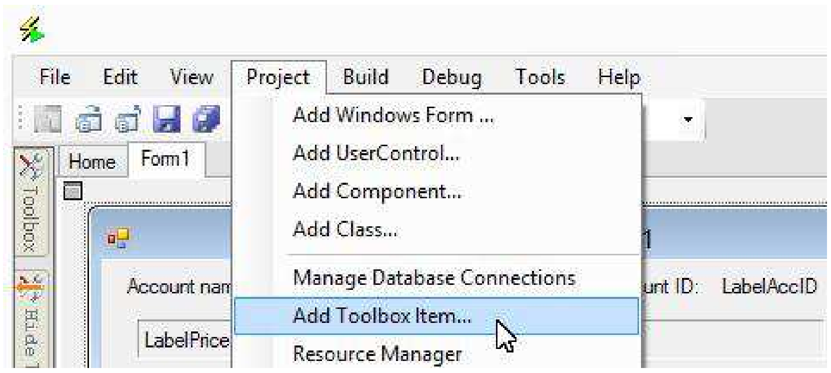
OANDA REST API is a software interface for currency trading (http://developer.oanda.com/). Since it supports Microsoft .Net Framework, you can use it in Limnor Studio. Usually you may get ideas of how to use a library in Limnor Studio by reading C# or VB.NET samples the library maker provides. If the library maker provides a .Net Framework interface then it will be easier to use the library in Limnor Studio by accessing the interface. For example, OANDA REST API provides a C# wrapper (https://github.com/oanda/CSharpLibAPISample). Compiling the wrapper into a DLL and load the DLL into your Limnor Studio projects, you can easily use the OANDA REST API. Below programming samples are described using the C# wrapper.

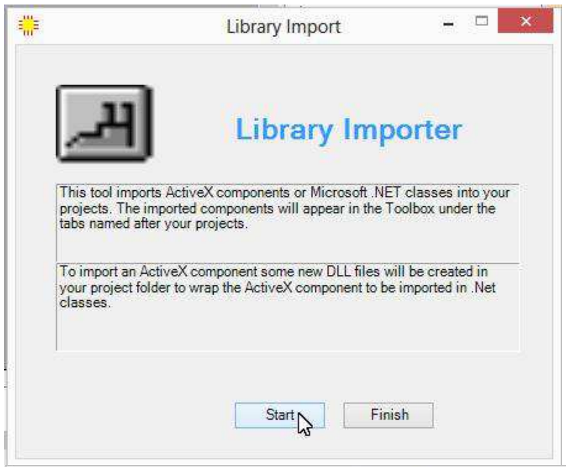
We have compiled the C# wrapper. The wrapper DLL can be found here http://www.limnor.com/support/RestAPI.zip.

The sample project used in the document is also contained in the above ZIP file.

Load Wrapper DLL

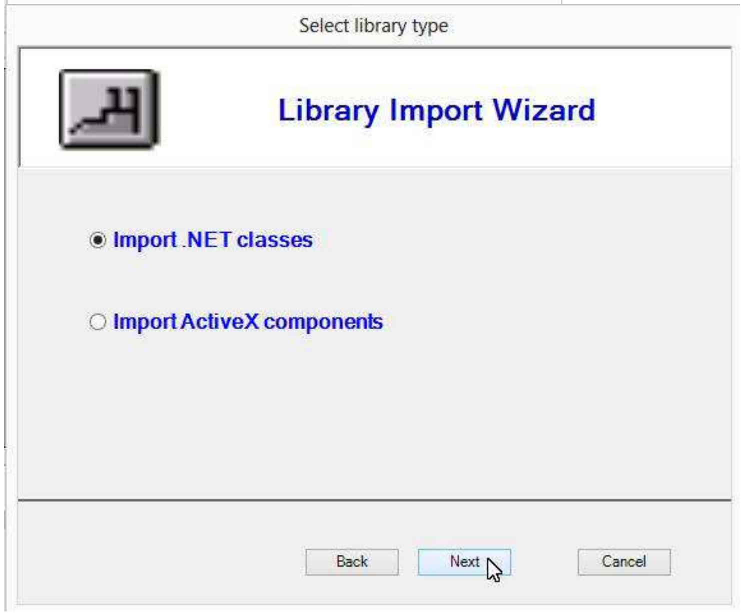
Create a Windows Form project. Add item to the Toolbox:





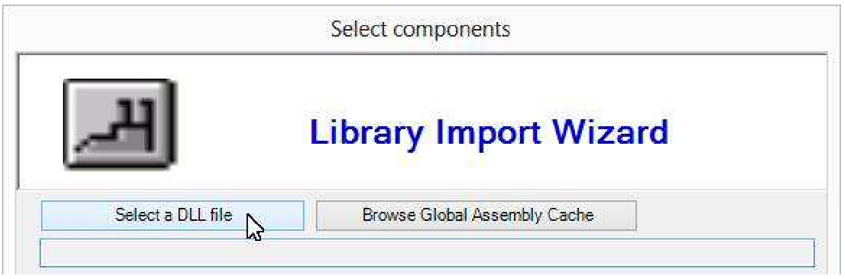
This tool imports ActiveX components or Microsoft .NET classes into your projects. The imported components will appear in the Toolbox under the tabs named after your projects.

To import an ActiveX component some new DLL files will be created in your project folder to wrap the ActiveX component to be imported in .Net classes.

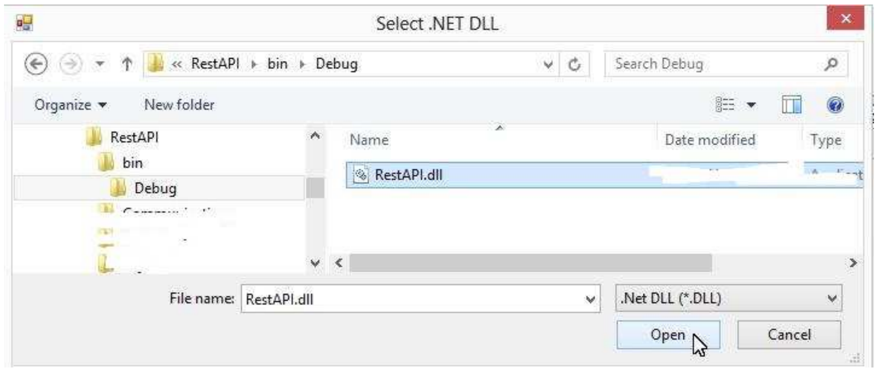


[● Import .NET classes]

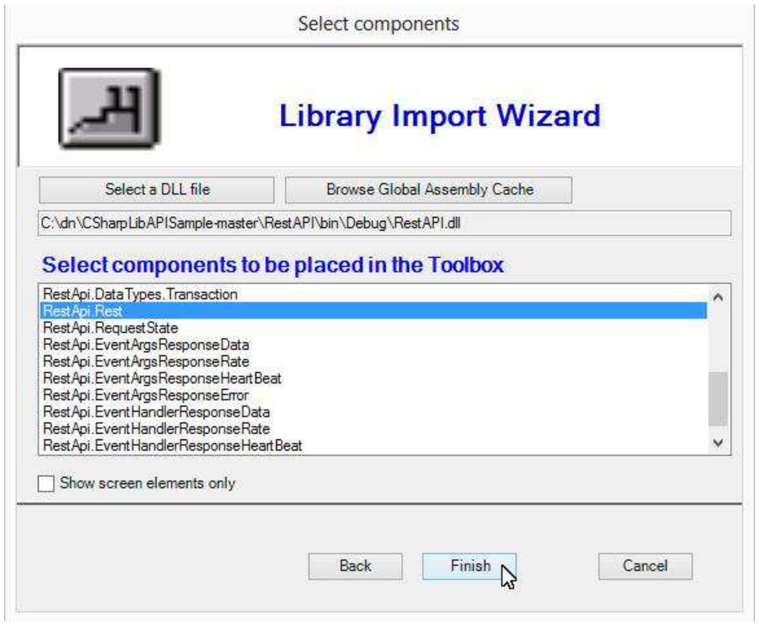
○ Import ActiveX components



Select the wrapper DLL:



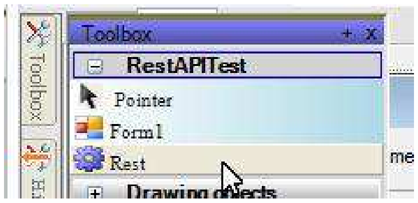
Types contained in the DLL are listed. According to the manufacturer, the class named Rest is the major interface class we are going to use. Select it and click Finish:


Select components
Library Import Wizard
Select a DLL file
Browse Global Assembly Cache
Select components to be placed in the Toolbox
Show screen elements only
Back
Finish
Cancel

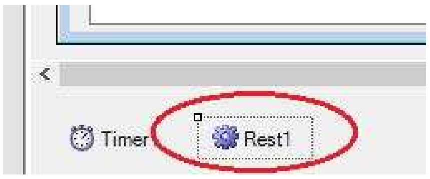
C:\dn\CSharpLibAPISample-master\RestAPI\bin\Debug\RestAPI.dll

RestApi.DataTypes.Transaction
RestApi.Rest
RestApi.RequestState
RestApi.EventArgsResponseData
RestApi.EventArgsResponseRate
RestApi.EventArgsResponseHeartBeat
RestApi.EventArgsResponseError
RestApi.EventHandlerResponseData
RestApi.EventHandlerResponseRate
RestApi.EventHandlerResponseHeartBeat

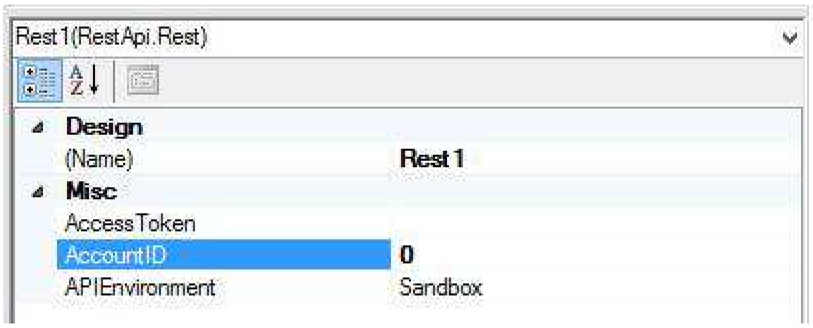
The class Rest appears in the Toolbox under the project:



Drop Rest to the form:

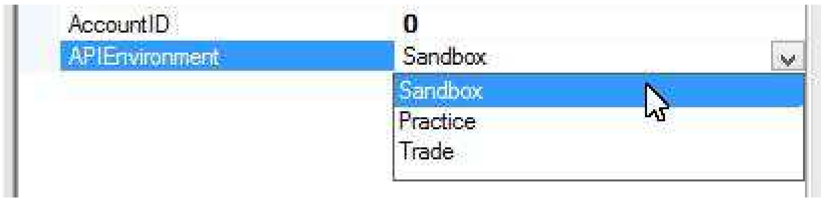


For easier programming, we modified the wrapper DLL by adding some properties to it:


Rest1(RestApi.Rest)

Design	
(Name)	**Rest1**
Misc	
AccessToken	
AccountID	**0**
APIEnvironment	Sandbox

- AccessToken – It is your password to the trading system. Keep it safe.
- AccountID – It is your account ID for the trading system.
- APIEnvironment – It is the environment to be used:

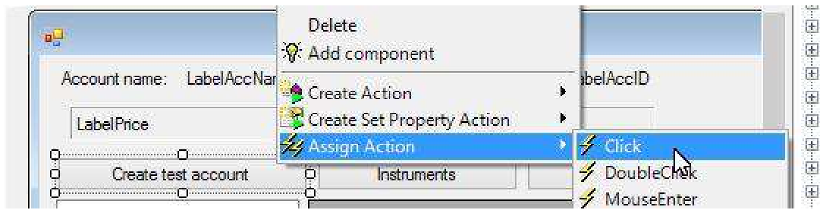


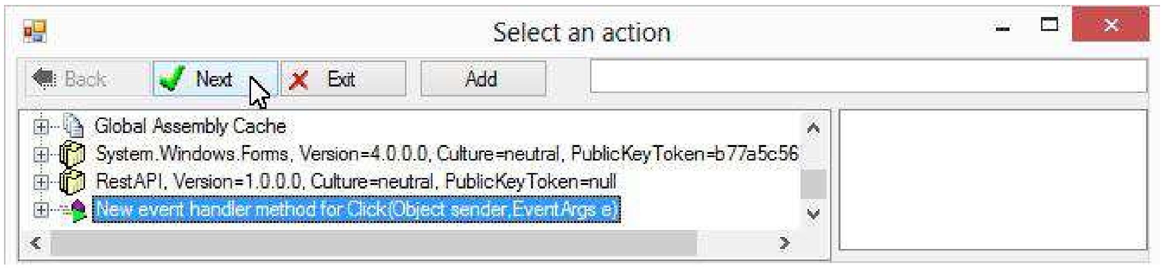
Sandbox and Practice are not for real trading. Trade is. See http://developer.oanda.com/docs/v1/guide/

By adding these properties, we do not have to pass them into actions, and thus simplify the programming.

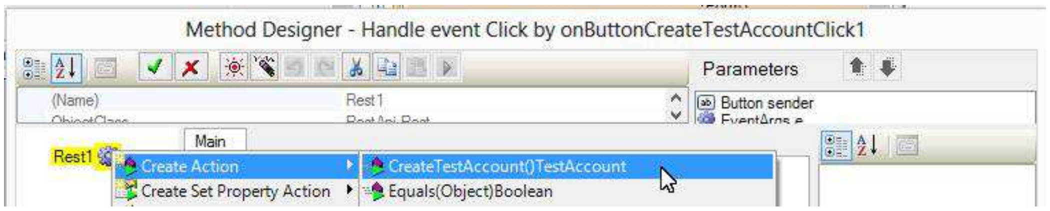
Create Sample Account

When using the Sandbox environment, we need to create a test trading account. Use a button to create a test account and show it on the screen:

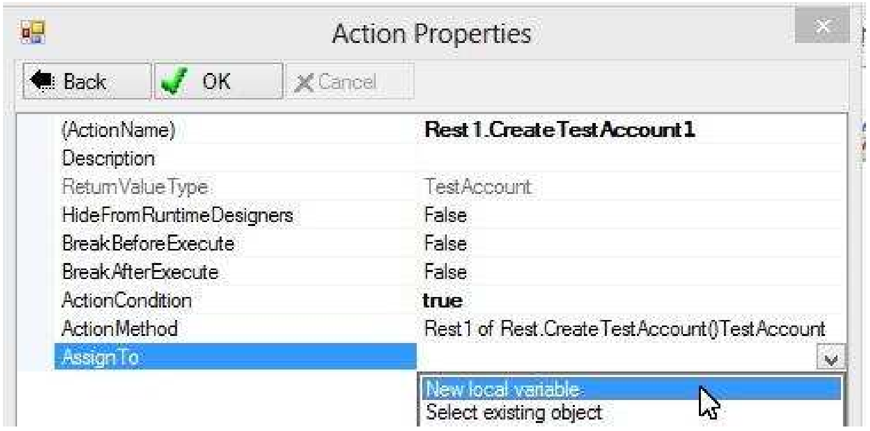



Select an action
Back
Next
Exit
Add
Global Assembly Cache
System.Windows.Forms, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c56
RestAPI, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
New event handler method for Click(Object sender,EventArgs e)

Create an action to create a test account:

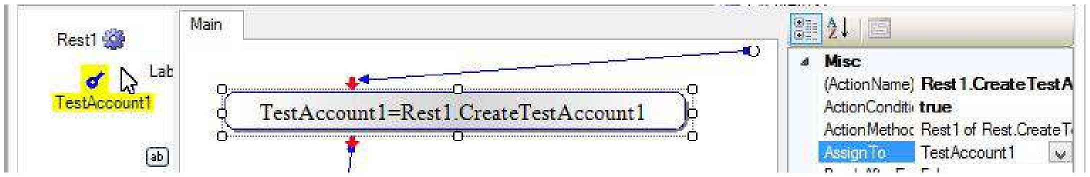

Method Designer - Handle event Click by onButtonCreateTestAccountClick1
Parameters
(Name) Rest1
Button sender
Main
Rest1
Create Action
Create Set Property Action
CreateTestAccount()TestAccount
Equals(Object)Boolean

Assign the action result to a new variable:



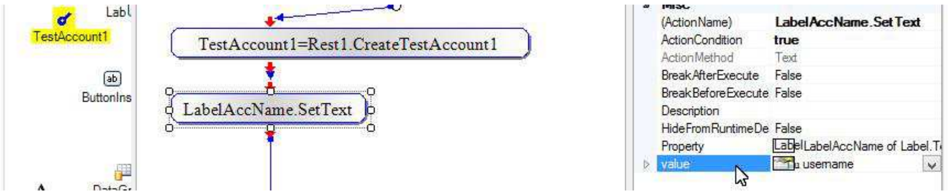
(ActionName)	**Rest1.CreateTestAccount1**
Description	
ReturnValueType	TestAccount
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	**true**
ActionMethod	Rest1 of Rest.CreateTestAccount()TestAccount
AssignTo	

Click OK, the action is created and a variable, TestAccount1, appears:



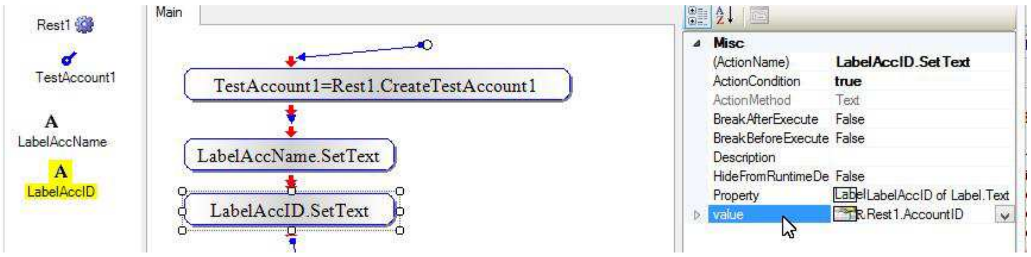
(ActionName)	**Rest1.CreateTestA**
ActionConditi	**true**
ActionMethoc	Rest1 of Rest.CreateT
AssignTo	TestAccount1

TestAccount1 has following properties: accountId, username, and password. We may show username in a label:



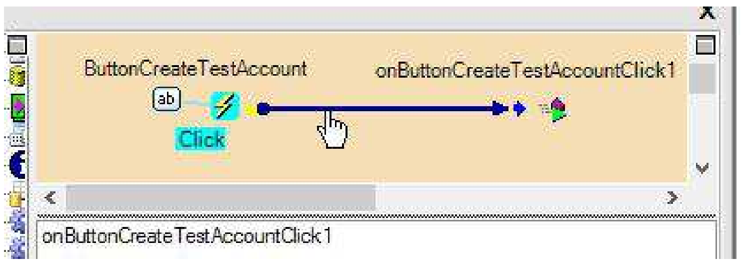
(ActionName)	**LabelAccName.SetText**
ActionCondition	**true**
ActionMethod	Text
BreakAfterExecute	False
BreakBeforeExecute	False
Description	
HideFromRuntimeDe	False
Property	LabelLabelAccName of Label.T
value	username

CreateTestAccount action also sets the AccountID property of Rest1 to the new account ID. We may display the new account ID on a label:

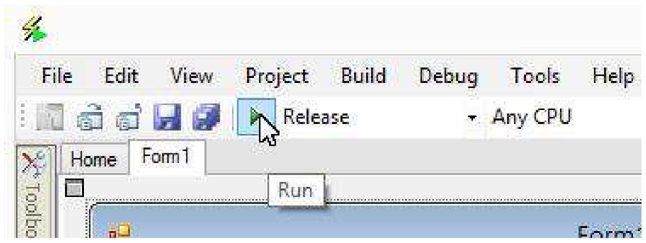


(ActionName)	**LabelAccID.SetText**
ActionCondition	**true**
ActionMethod	Text
BreakAfterExecute	False
BreakBeforeExecute	False
Description	
HideFromRuntimeDe	False
Property	LabelLabelAccID of Label.Text
value	R.Rest1.AccountID

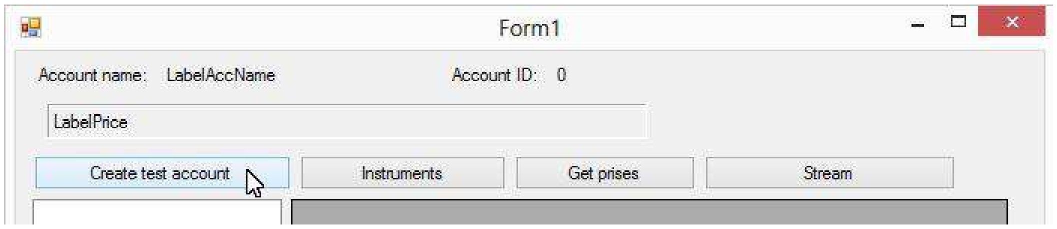
That is all for the button click event handler.


ButtonCreateTestAccount
ab
Click
onButtonCreateTestAccountClick1
onButtonCreateTestAccountClick1

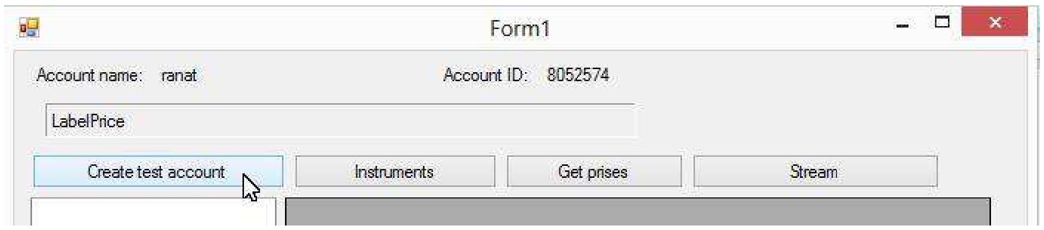
Let's compile and run the application:



The form appears. Click "Create test account":



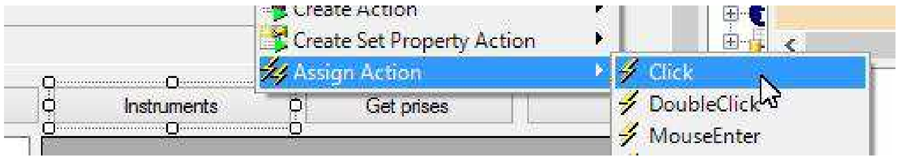
After a while, account name and account ID appear on the form:

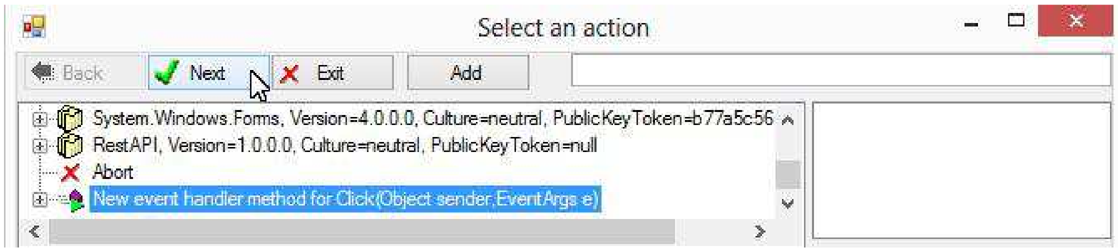


Get Instruments

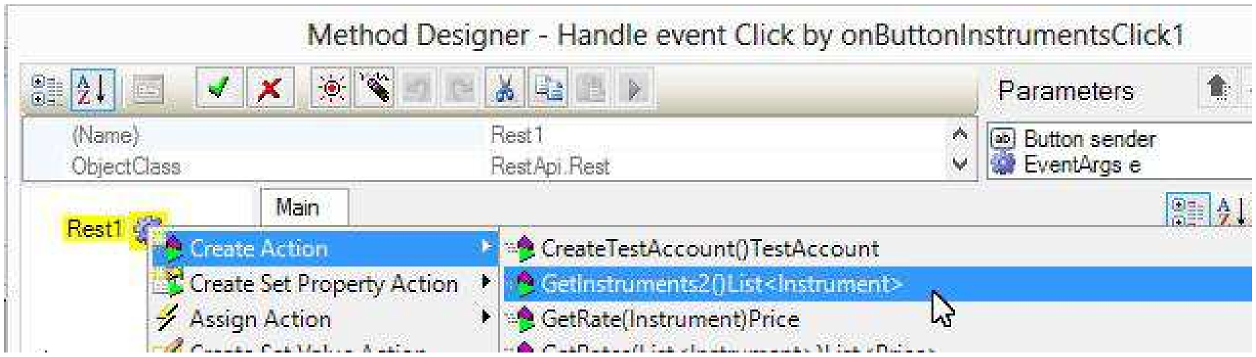
Use GetInstruments2

A GetInstruments action can be used to get all instruments. In this sample, we use a list box to show all the instruments.

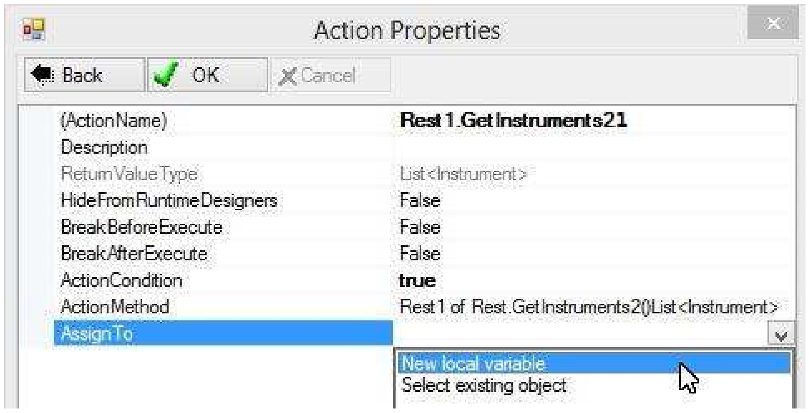



Select an action
Back
Next
Exit
Add
System.Windows.Forms, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c56
RestAPI, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
Abort
New event handler method for Click(Object sender,EventArgs e)

Create a GetInstrument2 action. Note that there is also a GetInstrument method which requires account ID. GetInstrument2 does not need account ID,

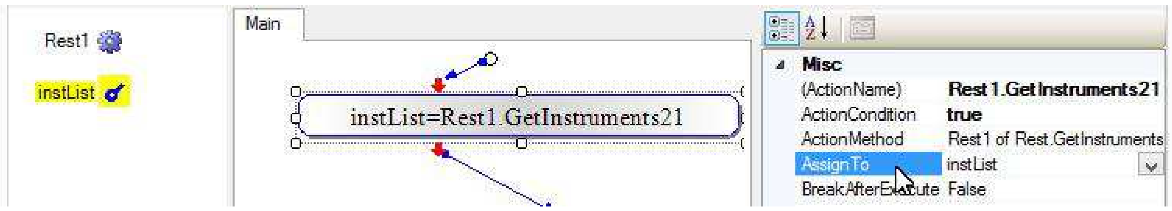

Method Designer - Handle event Click by onButtonInstrumentsClick1
Parameters
(Name)
Rest1
ObjectClass
RestApi.Rest
Button sender
EventArgs e
Main
Rest1
Create Action
Create Set Property Action
Assign Action
CreateTestAccount()TestAccount
GetInstruments2()List<Instrument>
GetRate(Instrument)Price

Assign the action result to a new variable:



(ActionName)	Rest1.GetInstruments21
Description	
ReturnValueType	List<Instrument>
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Rest1 of Rest.GetInstruments2()List<Instrument>
AssignTo	

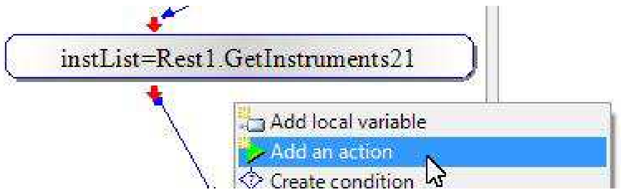
We named the new variable instList. Click OK. The action and the variable appear:

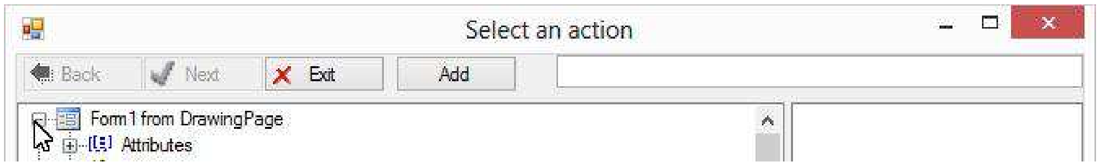


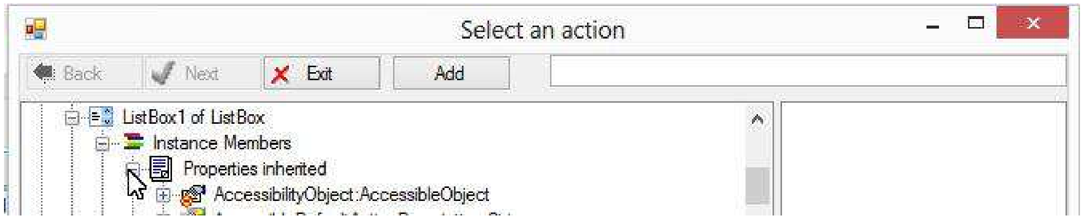
(ActionName)	Rest1.GetInstruments21
ActionCondition	true
ActionMethod	Rest1 of Rest.GetInstruments
AssignTo	instList
BreakAfterExecute	False

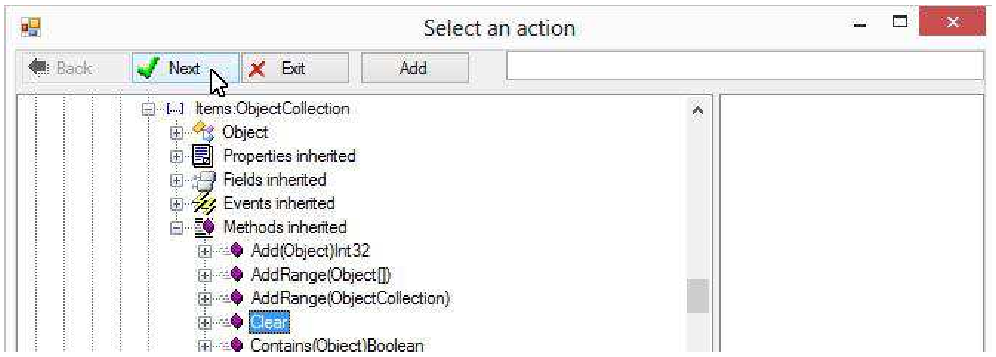
Remove list items

Variable instList is a list of all instruments. We want to display them in a list box. First, remove all existing items from the list box:



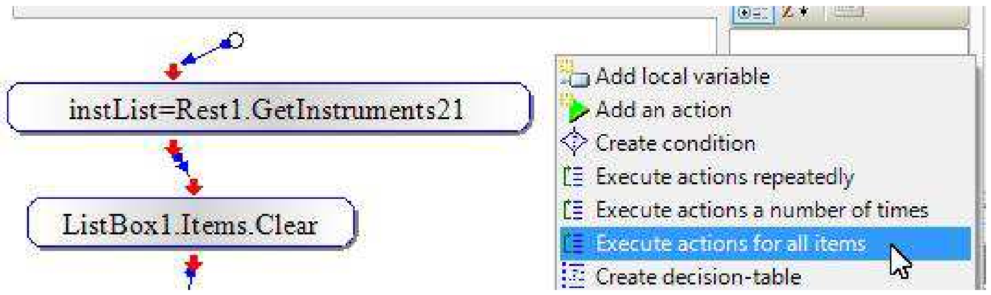




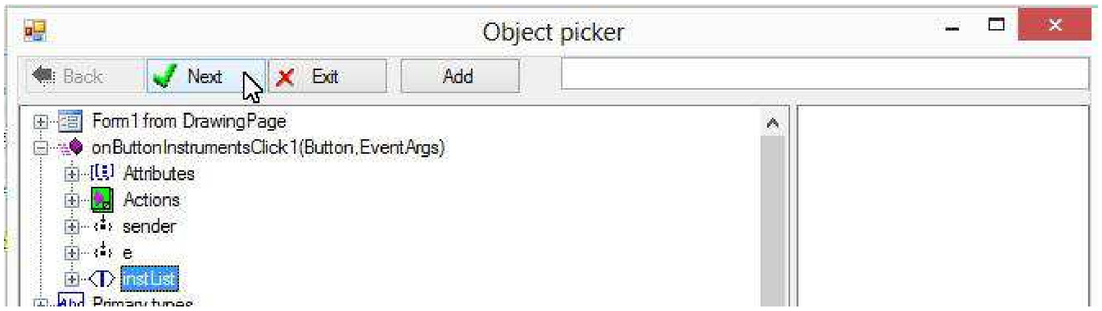


Go through instrument list

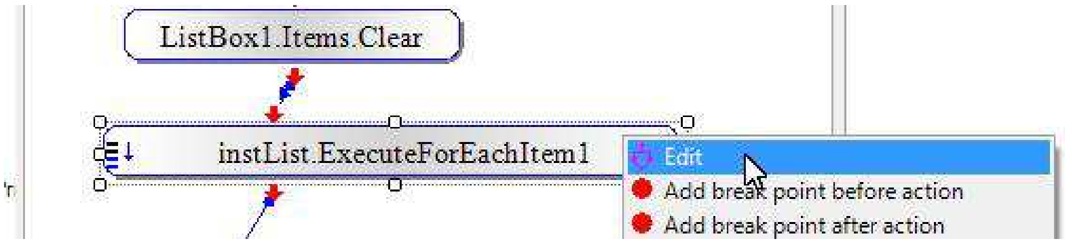
Add an "Execute actions for all items"


instList=Rest1.GetInstruments21
ListBox1.Items.Clear
Add local variable
Add an action
Create condition
Execute actions repeatedly
Execute actions a number of times
Execute actions for all items
Create decision-table

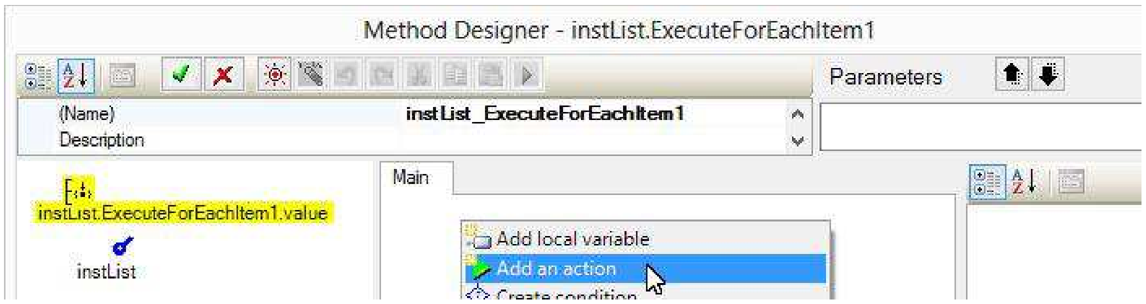
Choose variable instList:

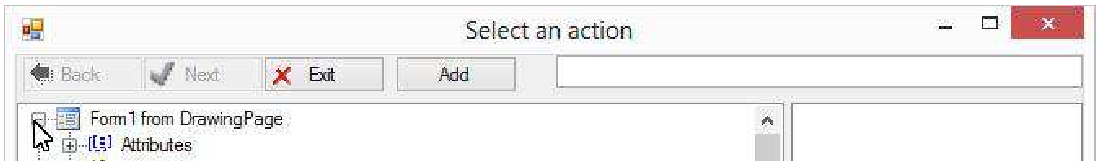


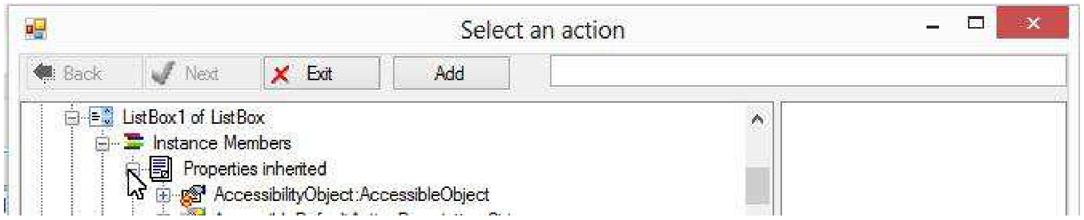
Edit it to add actions to it:

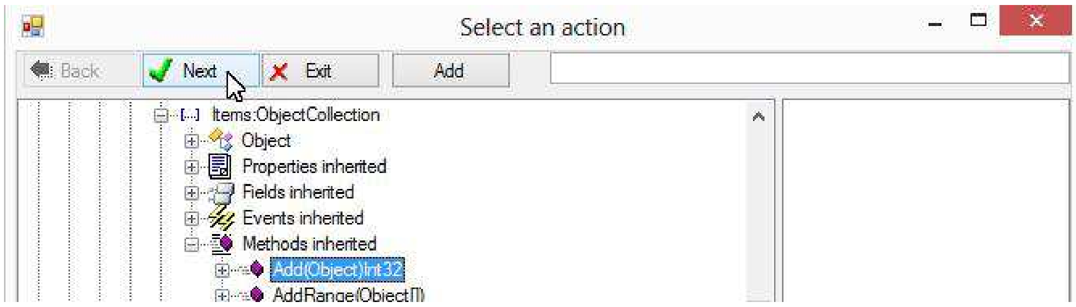

ListBox1.Items.Clear
instList.ExecuteForEachItem1
Edit
Add break point before action
Add break point after action

Note that the "value" variable represents the instrument to be processed. We add an action to add the instrument to the list box:

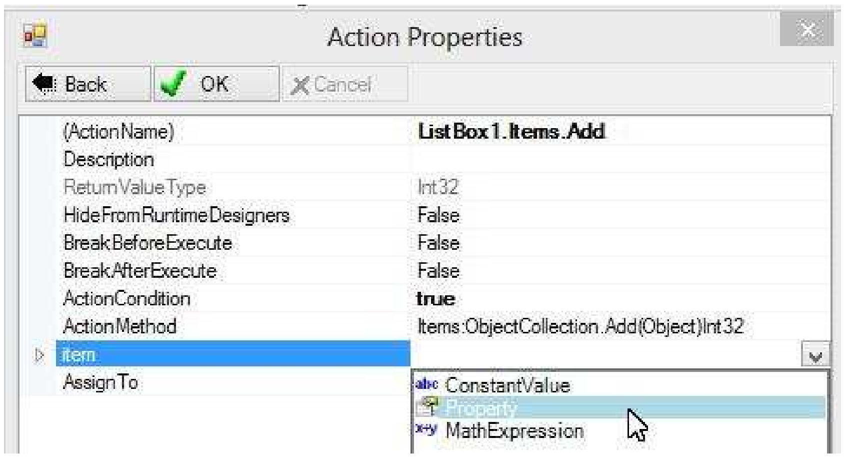

Method Designer - instList.ExecuteForEachItem1
Parameters
(Name)
instList_ExecuteForEachItem1
Description
Main
instList.ExecuteForEachItem1.value
instList
Add local variable
Add an action
Create condition



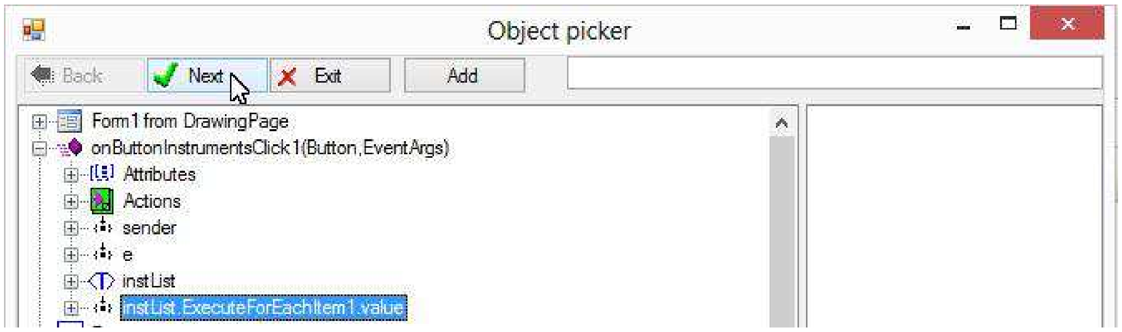




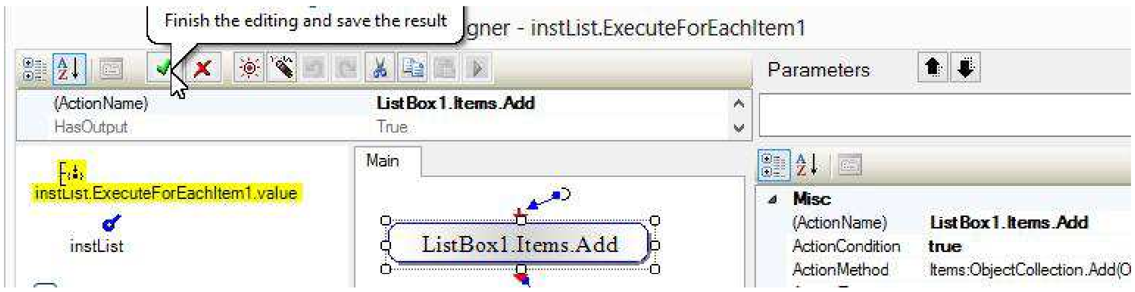
Use the “value” variable for the action:



(ActionName)	ListBox1.Items.Add
Description	
ReturnValueType	Int32
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Items:ObjectCollection.Add(Object)Int32
item	
AssignTo	



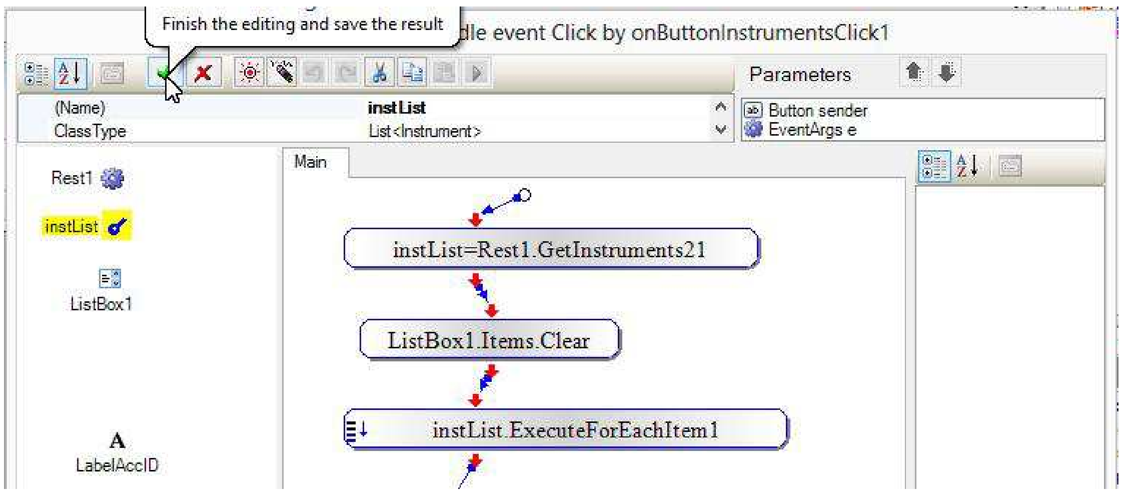
For this sample, we just need this one action.


Finish the editing and save the result
gner - instList.ExecuteForEachItem1
Parameters
Main
instList.ExecuteForEachItem1.value
instList
ListBox1.Items.Add
Misc

(ActionName)	ListBox1.Items.Add
HasOutput	True

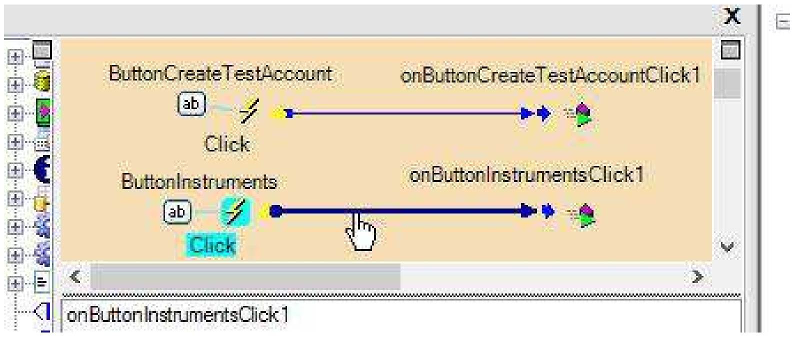
(ActionName)	ListBox1.Items.Add
ActionCondition	true
ActionMethod	Items:ObjectCollection.Add(O

That is all we need for the button event handler:


Finish the editing and save the result
dle event Click by onButtonInstrumentsClick1
Parameters
Button sender
EventArgs e
Main
Rest1
instList
ListBox1
LabelAccID

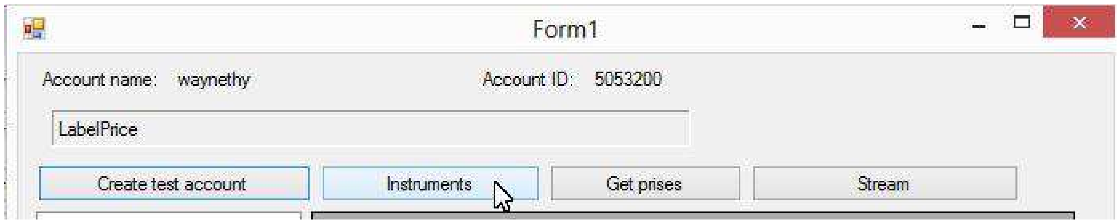
(Name)	instList
ClassType	List<Instrument>

```
instList=Rest1.GetInstruments21
ListBox1.Items.Clear
instList.ExecuteForEachItem1
```


ButtonCreateTestAccount
Click
onButtonCreateTestAccountClick1
ButtonInstruments
Click
onButtonInstrumentsClick1
onButtonInstrumentsClick1

Test

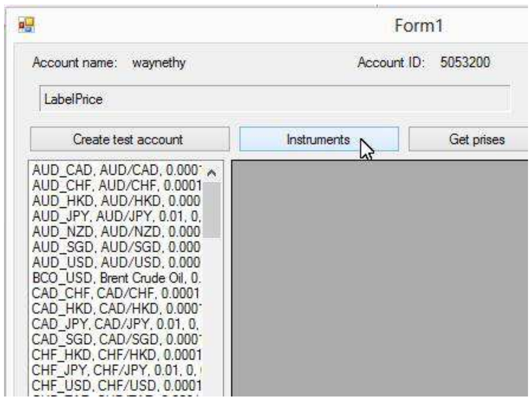
Compile and run the application. Click “Create test account” to create a test account. Then click “Instruments” button:



Account name: waynethy Account ID: 5053200

LabelPrice

All instruments appear in the list box:


Form1

Account name: waynethy Account ID: 5053200

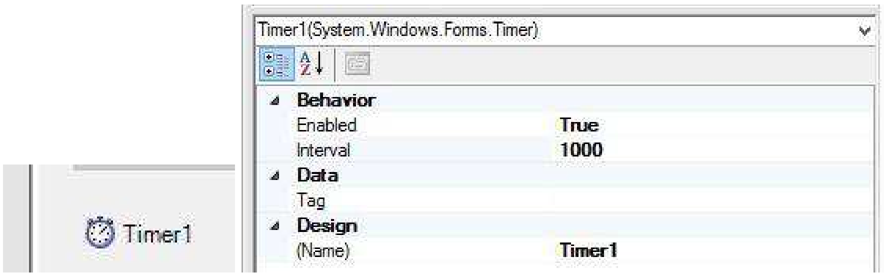
LabelPrice

```
AUD_CAD, AUD/CAD, 0.000
AUD_CHF, AUD/CHF, 0.0001
AUD_HKD, AUD/HKD, 0.000
AUD_JPY, AUD/JPY, 0.01, 0,
AUD_NZD, AUD/NZD, 0.000
AUD_SGD, AUD/SGD, 0.000
AUD_USD, AUD/USD, 0.000
BCO_USD, Brent Crude Oil, 0.
CAD_CHF, CAD/CHF, 0.0001
CAD_HKD, CAD/HKD, 0.000
CAD_JPY, CAD/JPY, 0.01, 0,
CAD_SGD, CAD/SGD, 0.000
CHF_HKD, CHF/HKD, 0.0001
CHF_JPY, CHF/JPY, 0.01, 0,
CHF_USD, CHF/USD, 0.0001
```

Get Ask/Bid

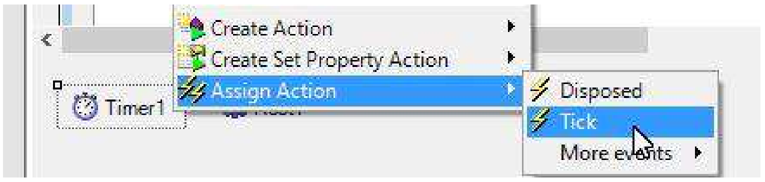
Use Timer

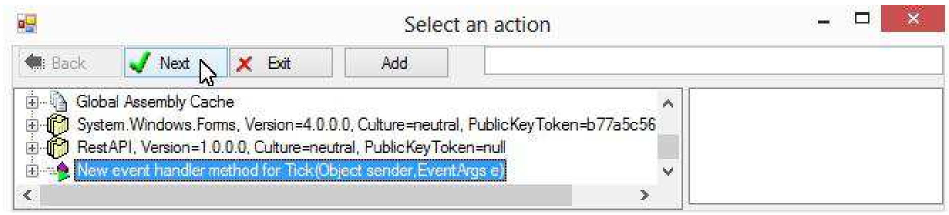
In every second we want to display ask/bid prices for the selected instrument. Add a timer to form.


Timer1(System.Windows.Forms.Timer)

Behavior	
Enabled	True
Interval	1000
Data	
Tag	
Design	
(Name)	Timer1

Handle Tick event to show ask/bid:

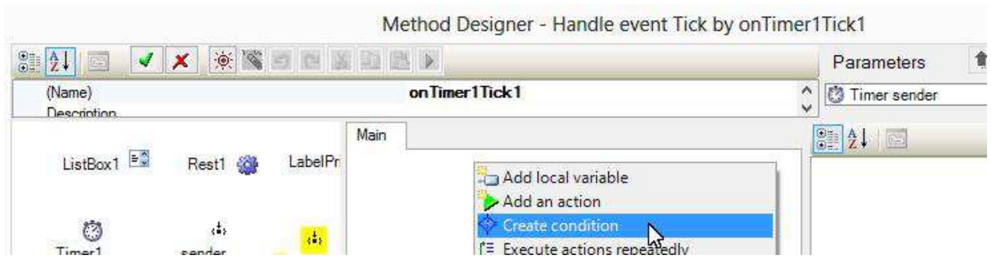



Select an action

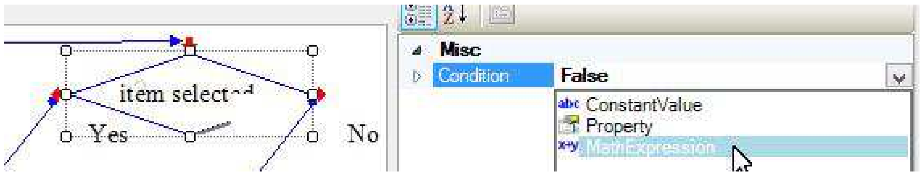
```
Global Assembly Cache
System.Windows.Forms, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c56
RestAPI, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
New event handler method for Tick(Object sender,EventArgs e)
```

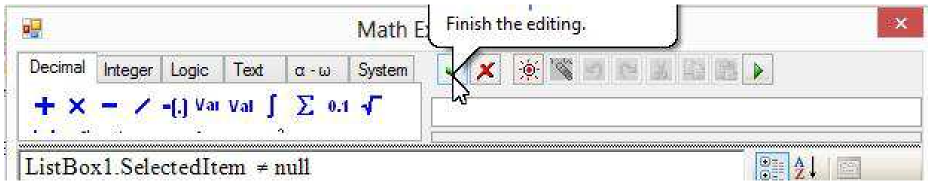
Check list selection

Add a condition to check whether a list item is selected:



Set condition that SelectedItem is not null:

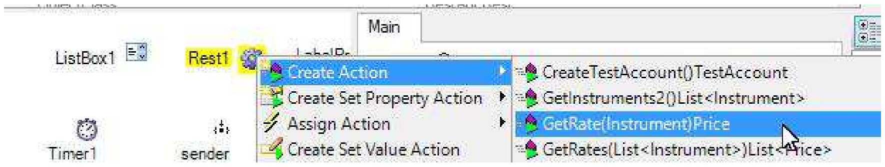




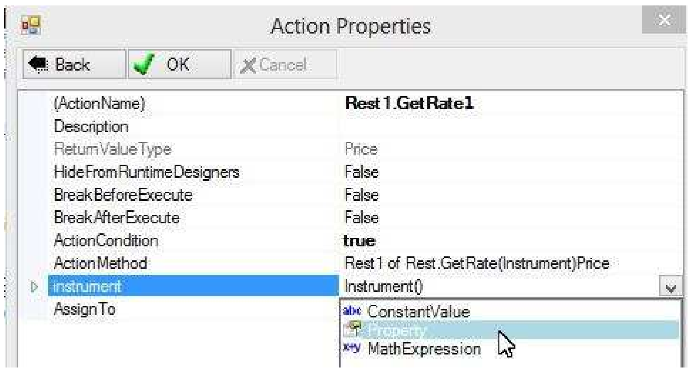
ListBox1.SelectedItem ≠ null

Get rate

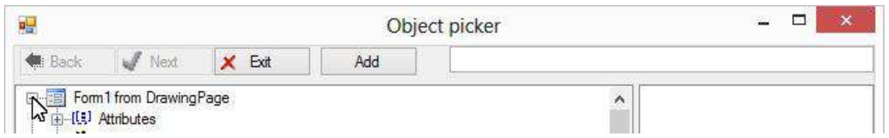
Create a GetRate action:

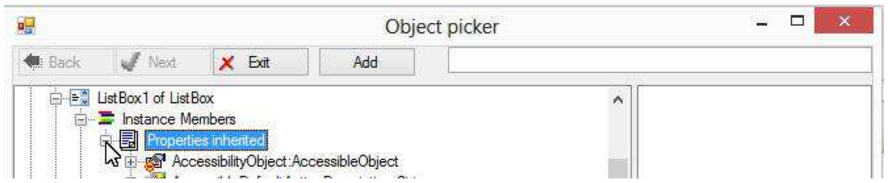

Main
ListBox1
Rest1
Timer1
sender
Create Action
Create Set Property Action
Assign Action
Create Set Value Action
CreateTestAccount()TestAccount
GetInstruments2()List<Instrument>
GetRate(Instrument)Price
GetRates(List<Instrument>)List<Price>

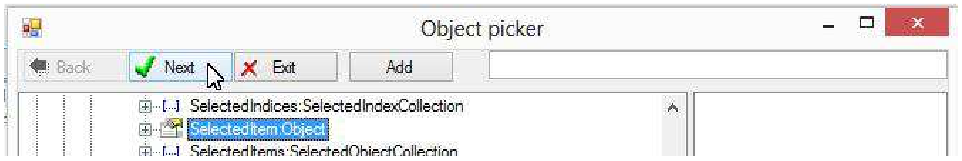
Pass SelectedItem to the action:



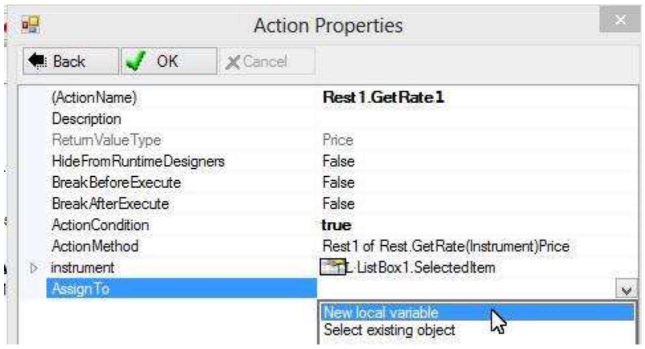
(ActionName)	Rest1.GetRate1
Description	
ReturnValueType	Price
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Rest1 of Rest.GetRate(Instrument)Price
instrument	Instrument()
AssignTo	





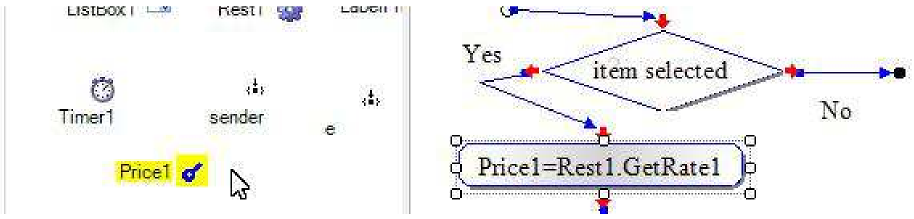

Object picker
Back
Next
Exit
Add
SelectedIndices:SelectedIndexCollection
SelectedItem:Object
SelectedItems:SelectedObjectCollection

Assign the action result to a new variable:



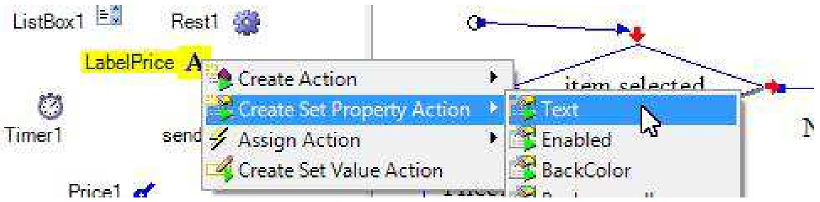
(ActionName)	Rest1.GetRate1
Description	
ReturnValueType	Price
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Rest1 of Rest.GetRate(Instrument)Price
instrument	ListBox1.SelectedItem
AssignTo	

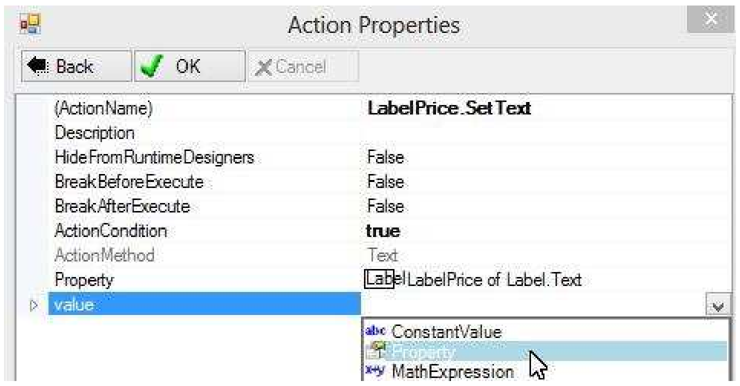
A new variable and an action appear:



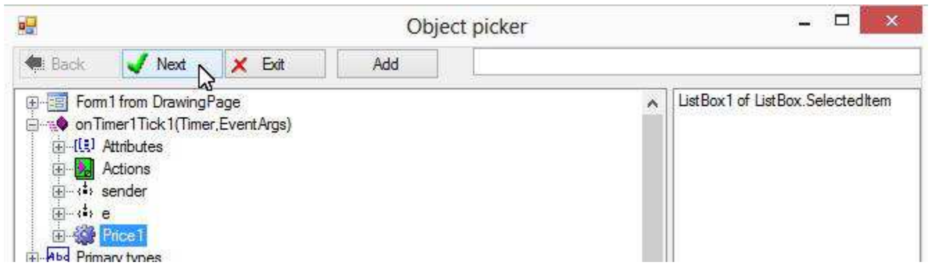
Show rate

Variable Price1 has following properties: instrument, ask, bid, time. We may show Price1 on a label:

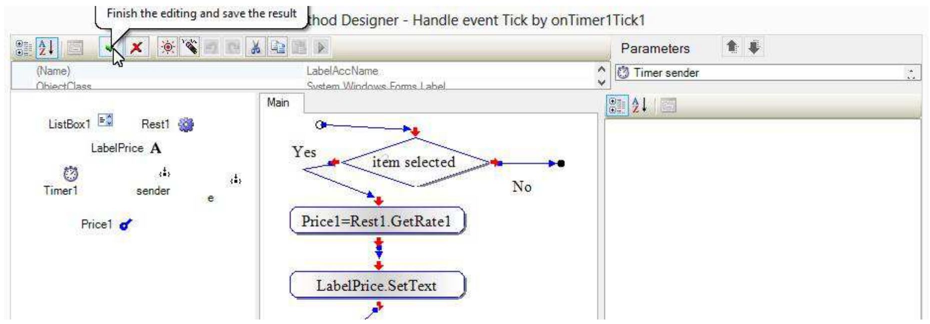




(ActionName)	**LabelPrice.SetText**
Description	
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	**true**
ActionMethod	Text
Property	Label LabelPrice of Label.Text
value	

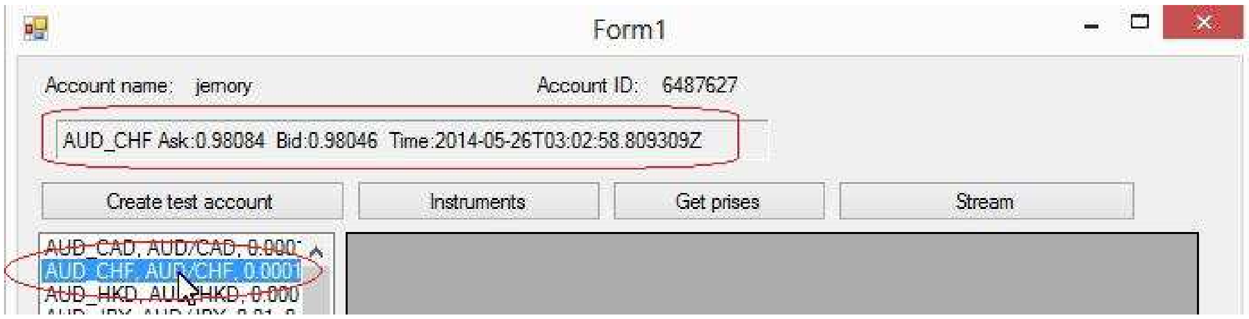


This is our timer Tick event handler:



Test

Compile and run the application. Click “Create test account”. Click “Instruments”. Select an item in the list. The prices for the instrument appear:



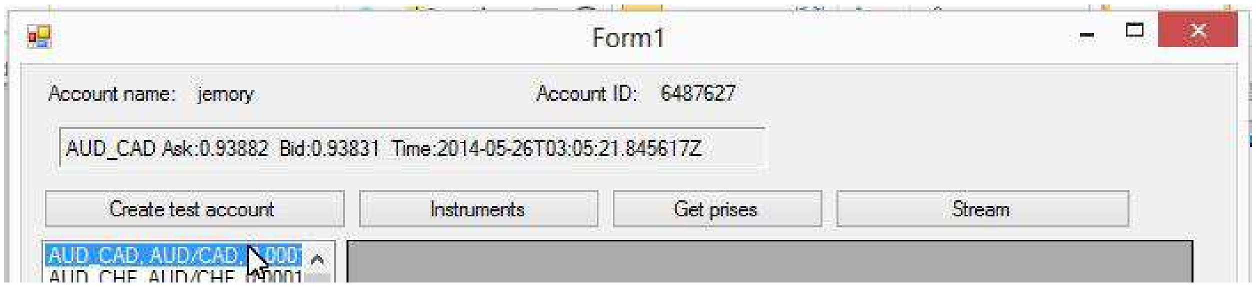
Account name: jemory Account ID: 6487627

AUD_CHF Ask:0.98084 Bid:0.98046 Time:2014-05-26T03:02:58.809309Z

AUD_CAD, AUD/CAD, 0.000
AUD_CHF, AUD/CHF, 0.0001
AUD_HKD, AUD/HKD, 0.000

The information for the selected instrument changes every second. The information may take longer than one second because connecting to internet takes time.

Click another instrument. The corresponding information appears:



Account name: jemory   Account ID: 6487627

AUD_CAD Ask:0.93882 Bid:0.93831 Time:2014-05-26T03:05:21.845617Z

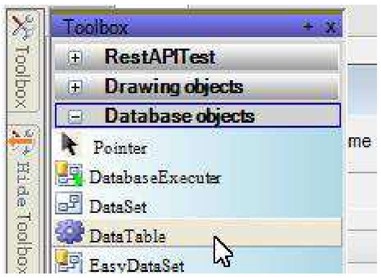
AUD_CAD, AUD/CAD, 0.0001
AUD_CHF, AUD/CHF, 0.0001

Show all prices

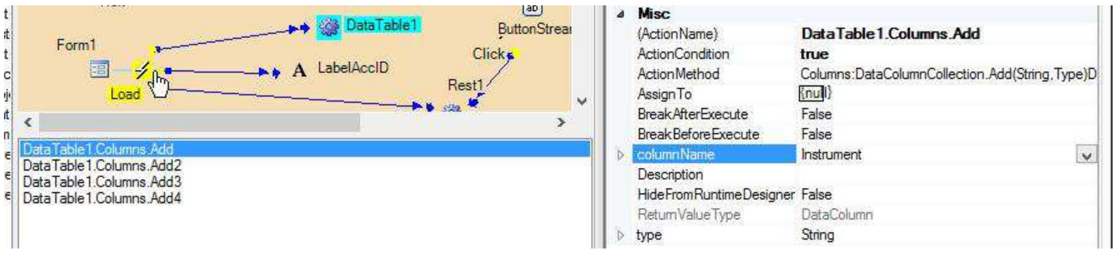
The above example gets information for one instrument. GetRates can be used to get information for several instruments.

Prepare DataTable

We are going to use a DataTable to hold prices returned from a GetRates action. Drop DataTable to the form:



At form Load event, create 4 actions to add 4 columns to DataTable:

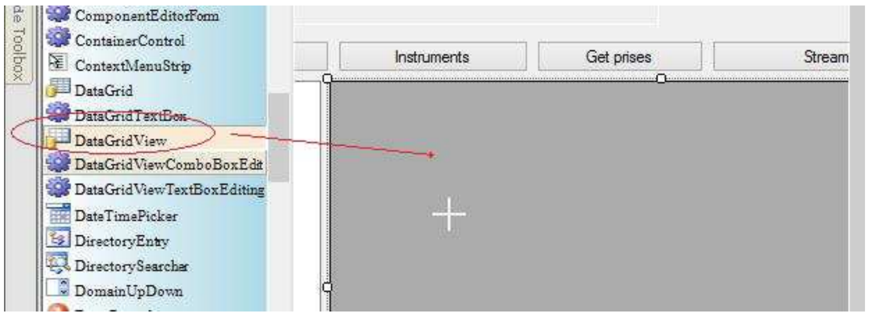


DataTable1.Columns.Add
DataTable1.Columns.Add2
DataTable1.Columns.Add3
DataTable1.Columns.Add4

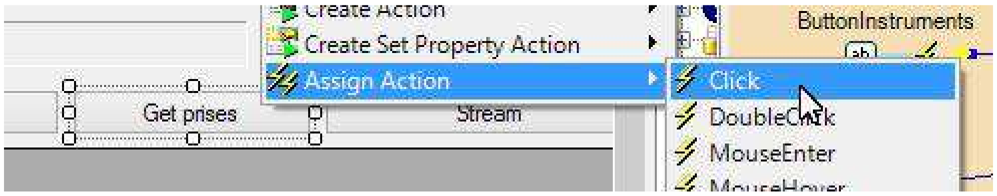
Misc	
(ActionName)	DataTable1.Columns.Add
ActionCondition	true
ActionMethod	Columns:DataColumnCollection.Add(String,Type)D
AssignTo	{null}
BreakAfterExecute	False
BreakBeforeExecute	False
columnName	Instrument
Description	
HideFromRuntimeDesigner	False
ReturnValueType	DataColumn
type	String

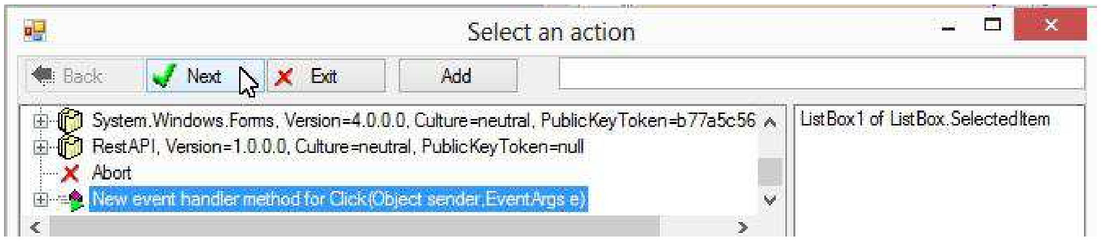
Add DataGridView

We are going to use a DataGridView to data:



Handle button click



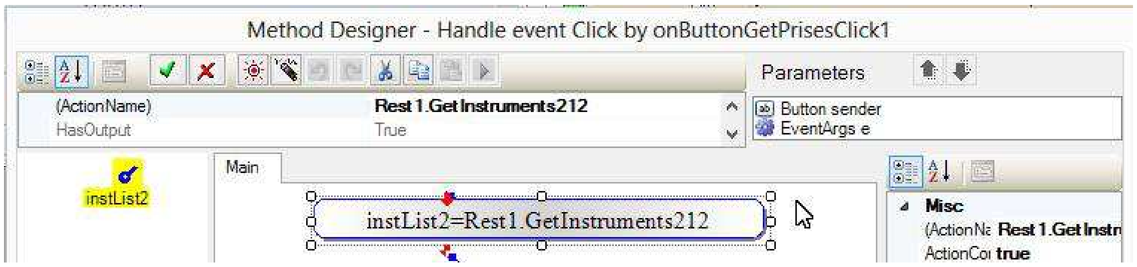


System.Windows.Forms, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c56
RestAPI, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
Abort
New event handler method for Click(Object sender,EventArgs e)

ListBox1 of ListBox.SelectedItem

Get all instruments

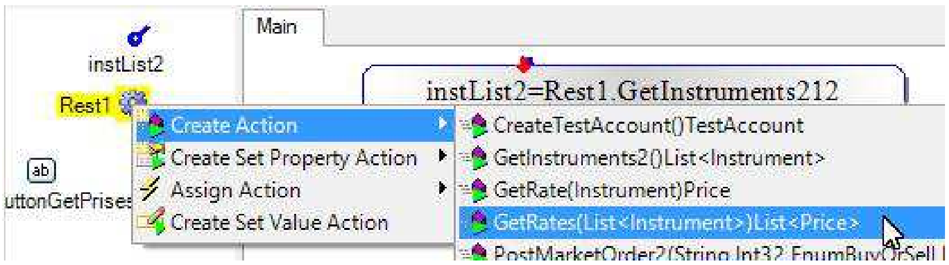
We have done it before in section “Get Instruments”. A GetInstrument2 action generates a variable intList2 which is a list of instruments.


Method Designer - Handle event Click by onButtonGetPrisesClick1
Parameters
Button sender
EventArgs e
instList2
Main
instList2=Rest1.GetInstruments212
Misc

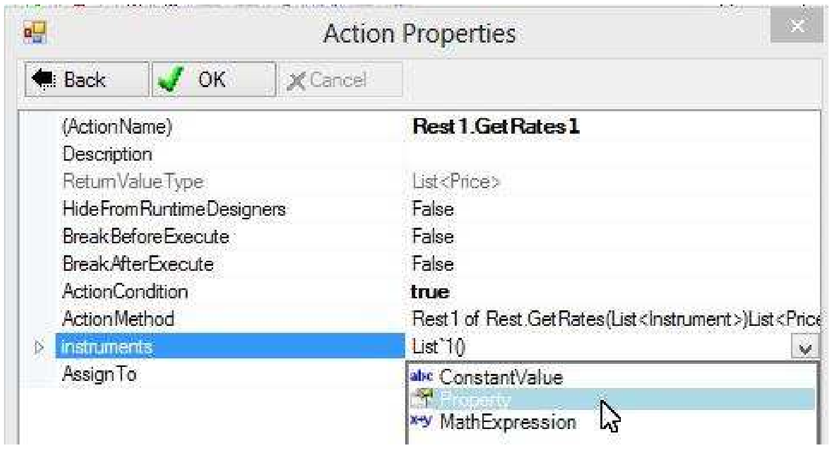
(ActionName)	Rest1.GetInstruments212
HasOutput	True

Get prices

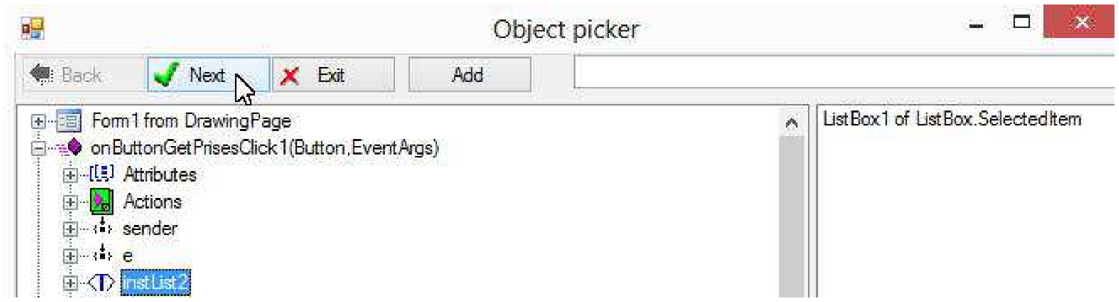
We may pass instList2 to function GetRates to get price list.


instList2
Rest1
Main
instList2=Rest1.GetInstruments212
uttonGetPrises
Create Action
Create Set Property Action
Assign Action
Create Set Value Action
CreateTestAccount()TestAccount
GetInstruments2()List<Instrument>
GetRate(Instrument)Price
GetRates(List<Instrument>)List<Price>
PostMarketOrder2(String,Int32,EnumBuyOrSell

Pass instList2 to the action:

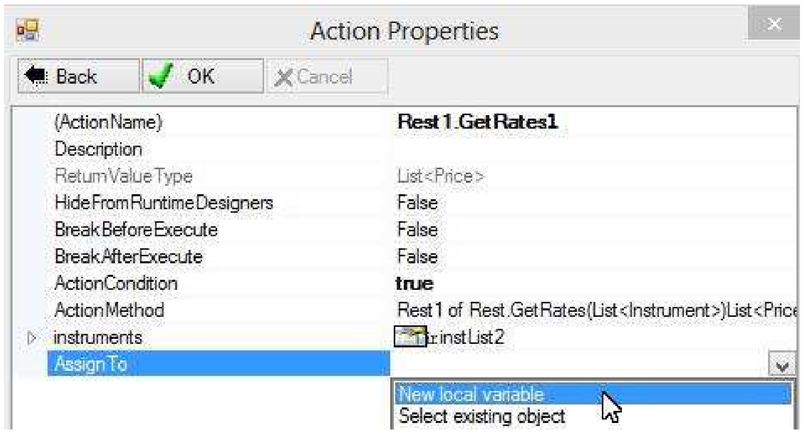

Action Properties

(ActionName)	**Rest1.GetRates1**
Description	
ReturnValueType	List<Price>
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	**true**
ActionMethod	Rest1 of Rest.GetRates(List<Instrument>)List<Price
instruments	List`1()
AssignTo	


Object picker

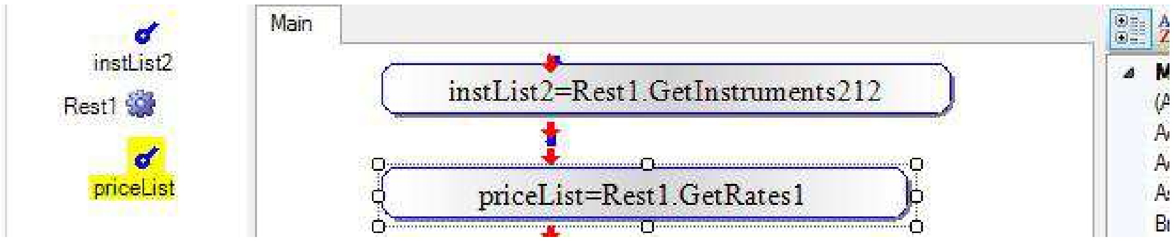
Back
Next
Exit
Add
Form1 from DrawingPage
onButtonGetPrisesClick1(Button,EventArgs)
Attributes
Actions
sender
e
instList2
ListBox1 of ListBox.SelectedItem

Assign the action result to a new variable:


Action Properties

(ActionName)	**Rest1.GetRates1**
Description	
ReturnValueType	List<Price>
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	**true**
ActionMethod	Rest1 of Rest.GetRates(List<Instrument>)List<Price
instruments	:instList2
AssignTo	

The action and a new variable appear:



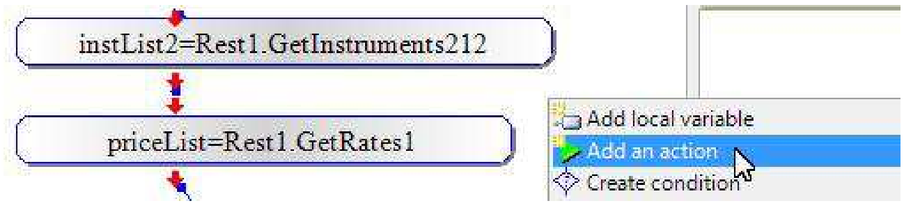
Variable priceList is a list of prices. Each item in the list is a Price object. Each Price object has following properties: instrument, ask, bid, and time.

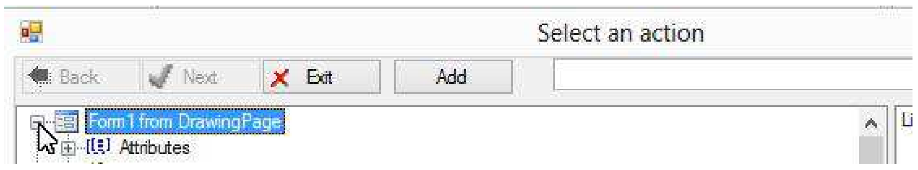
Populate DataTable

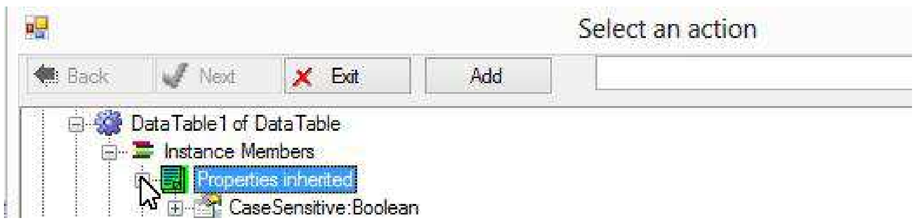
We'll populate DataTable1 with data from priceList.

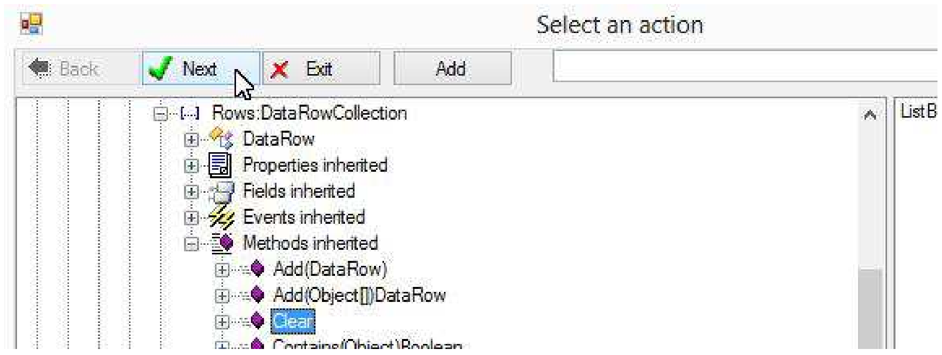
Remove existing rows

First we create an action to remove all rows from the table:



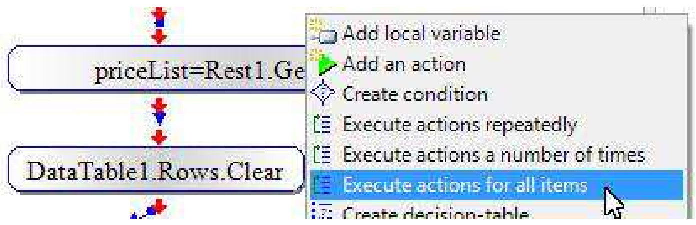




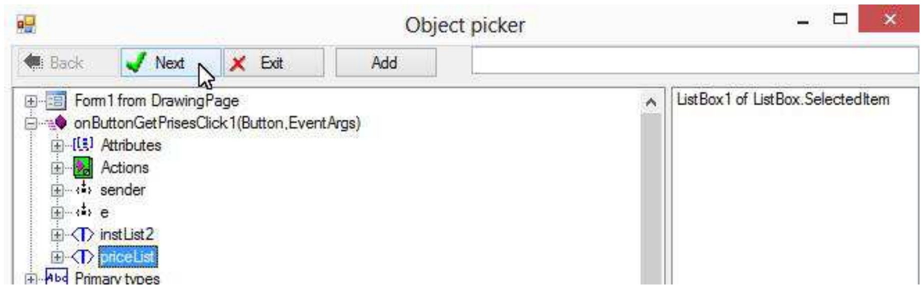


Go through prices

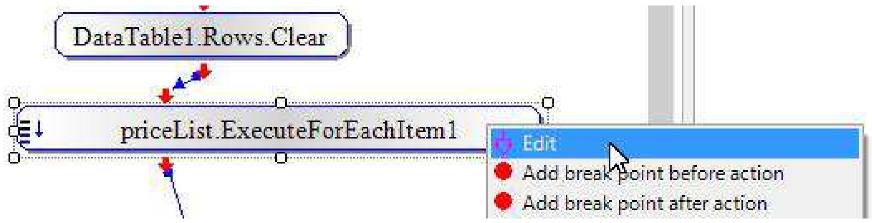
Add an action of "Execute actions for all items":


priceList=Rest1.Ge
DataTable1.Rows.Clear
Add local variable
Add an action
Create condition
Execute actions repeatedly
Execute actions a number of times
Execute actions for all items

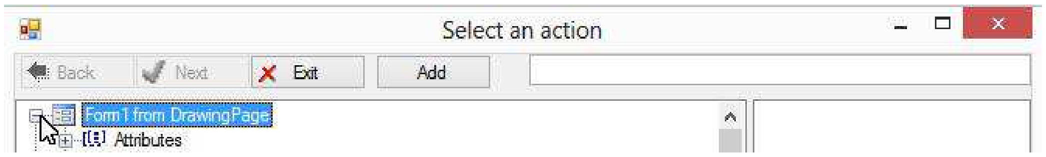
Select variable priceList:

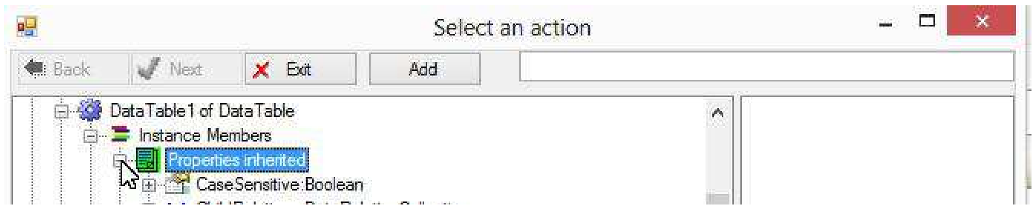


Edit the action to add actions:

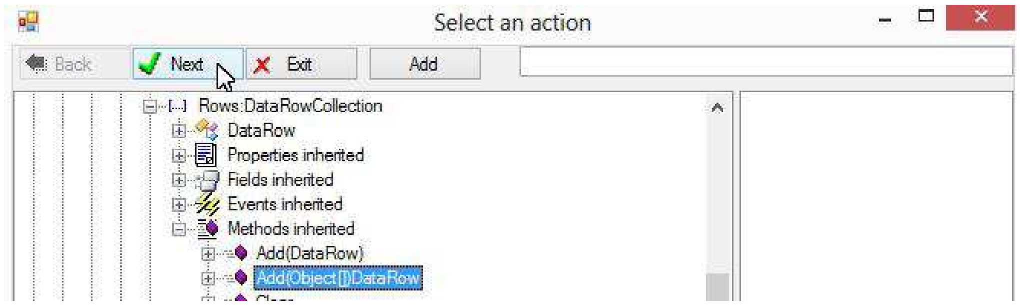

DataTable1.Rows.Clear
priceList.ExecuteForEachItem1
Edit
Add break point before action
Add break point after action

Note that the "value" variable represents the Price object to be processed. Add an action to add a row using properties from this "value" variable:

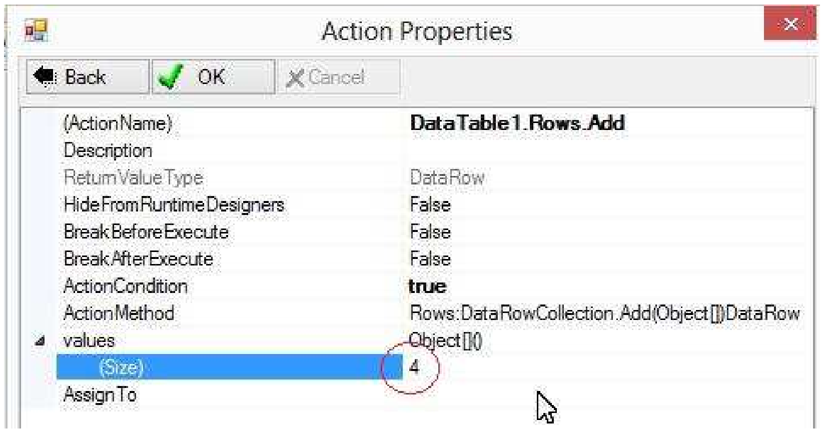




Select Add method of the Rows of DataTable1:

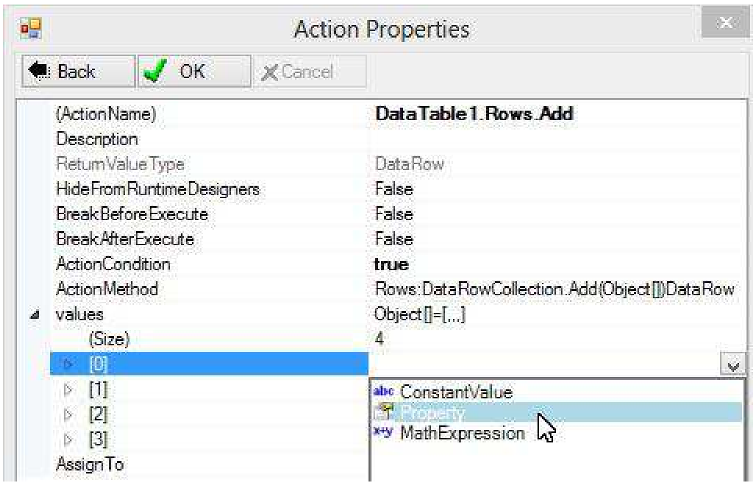


DataTable1 has 4 columns: instrument, ask, bid, and time. So, set the size of "values" to 4:

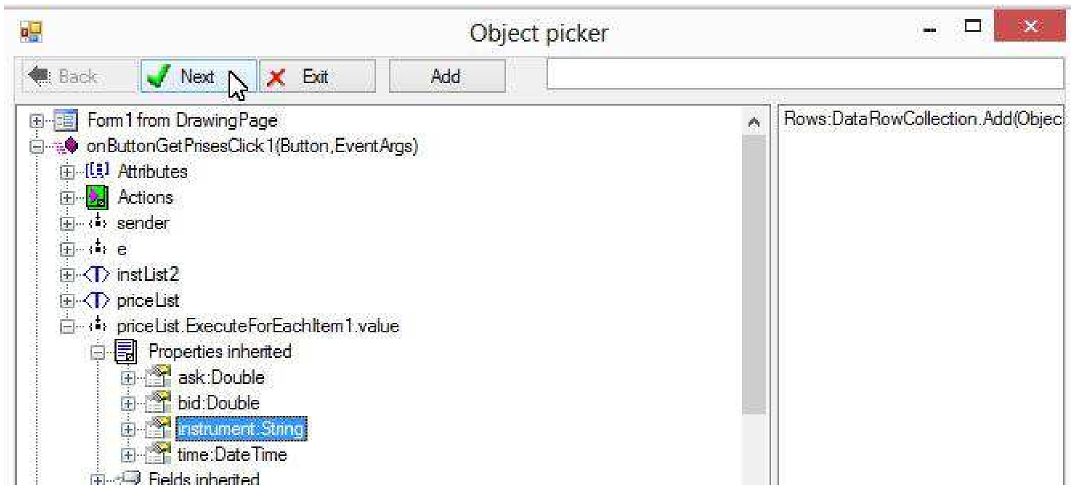


(ActionName)	DataTable1.Rows.Add
Description	
ReturnValueType	DataRow
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Rows:DataRowCollection.Add(Object[])DataRow
values	Object[]()
(Size)	4
AssignTo	

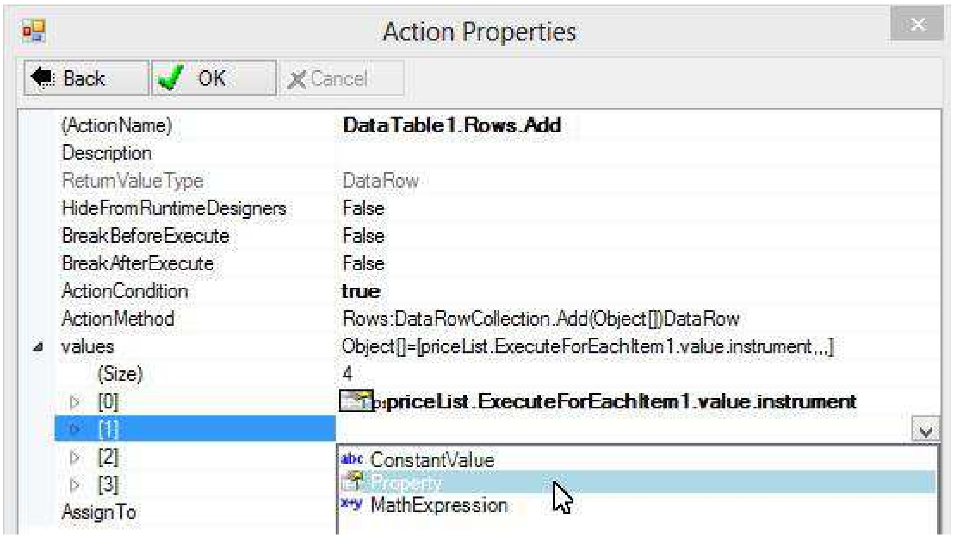
Set the first value to property instrument of the Price:



(ActionName)	DataTable1.Rows.Add
Description	
ReturnValueType	DataRow
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Rows:DataRowCollection.Add(Object[])DataRow
values	Object[]=[...]
(Size)	4
[0]	
[1]	
[2]	
[3]	
AssignTo	

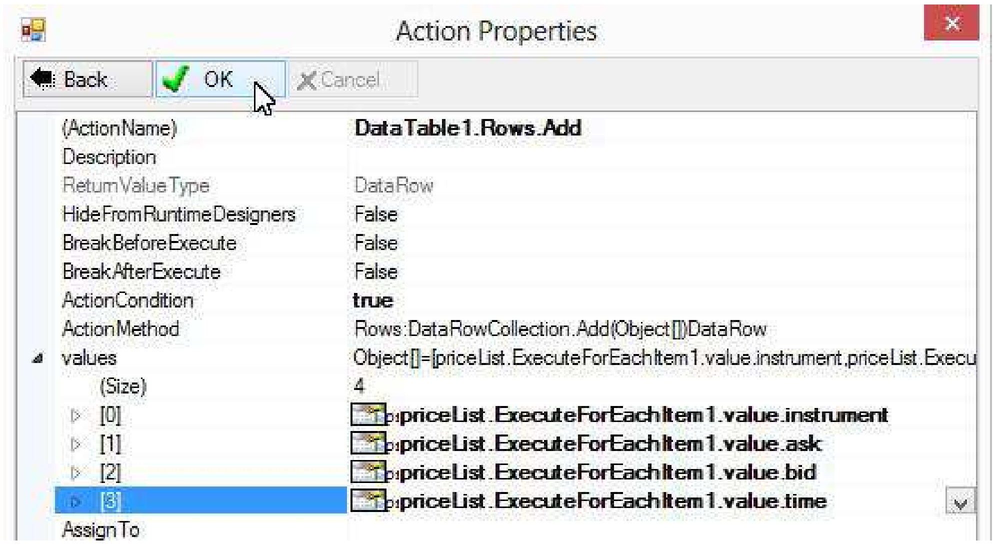


Set the second value to property ask:



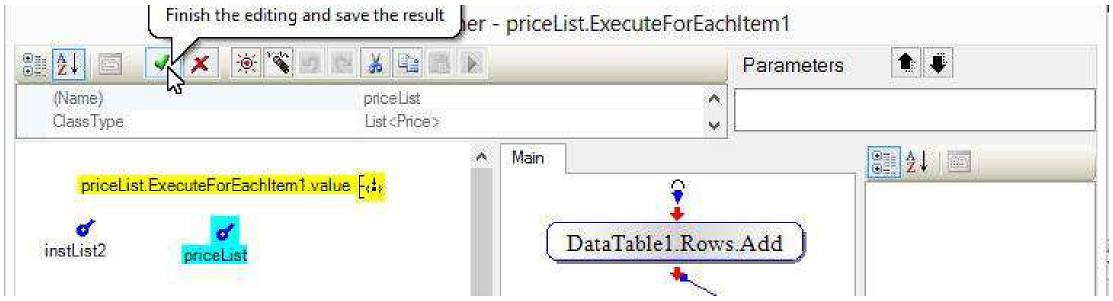
(ActionName)	DataTable1.Rows.Add
Description	
ReturnValueType	DataRow
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Rows:DataRowCollection.Add(Object[])DataRow
values	Object[]=[priceList.ExecuteForEachItem1.value.instrument,..]
(Size)	4
[0]	p:priceList.ExecuteForEachItem1.value.instrument
[1]	
[2]	
[3]	
AssignTo	

The third value is “bid” and the 4th value is “time”:



(ActionName)	DataTable1.Rows.Add
Description	
ReturnValueType	DataRow
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Rows:DataRowCollection.Add(Object[])DataRow
values	Object[]=[priceList.ExecuteForEachItem1.value.instrument,priceList.Execu
(Size)	4
[0]	p:priceList.ExecuteForEachItem1.value.instrument
[1]	p:priceList.ExecuteForEachItem1.value.ask
[2]	p:priceList.ExecuteForEachItem1.value.bid
[3]	p:priceList.ExecuteForEachItem1.value.time
AssignTo	

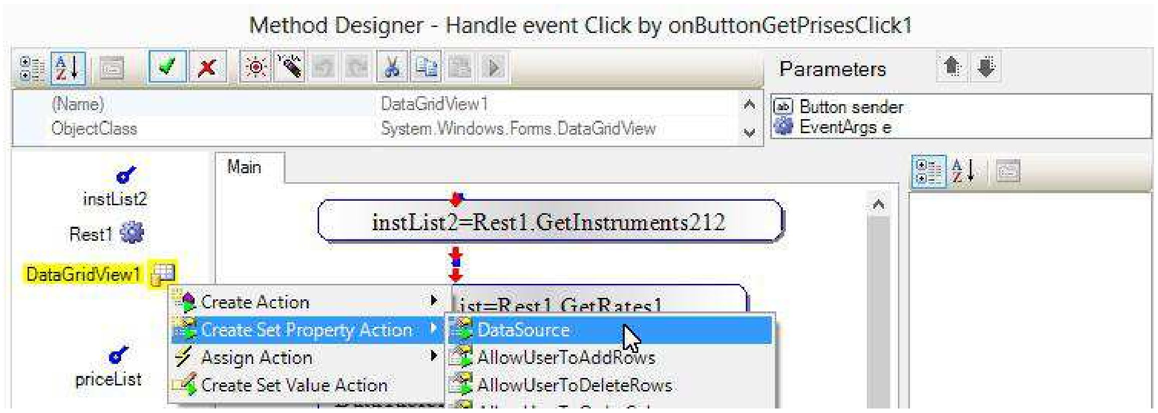
For this sample, we just need this one action:

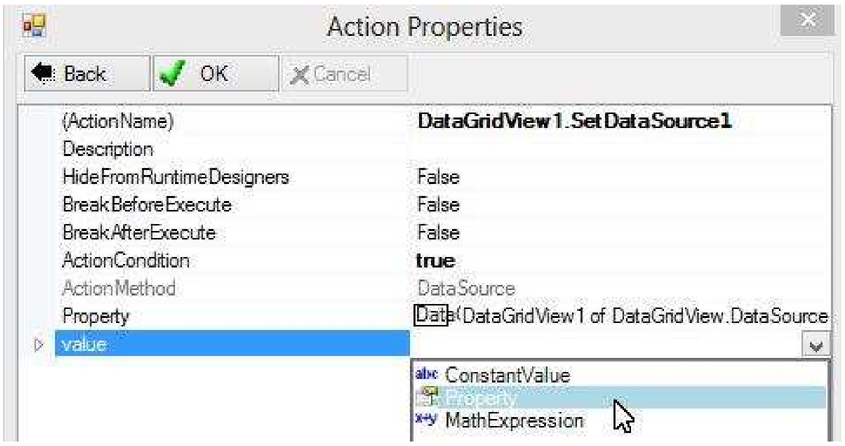

Finish the editing and save the result
ier - priceList.ExecuteForEachItem1
Parameters
priceList.ExecuteForEachItem1.value
instList2
priceList
Main
DataTable1.Rows.Add

(Name)	priceList
ClassType	List<Price>

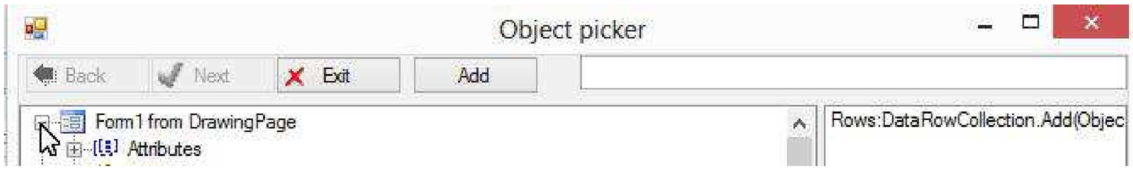
Data-binding of DataGridView

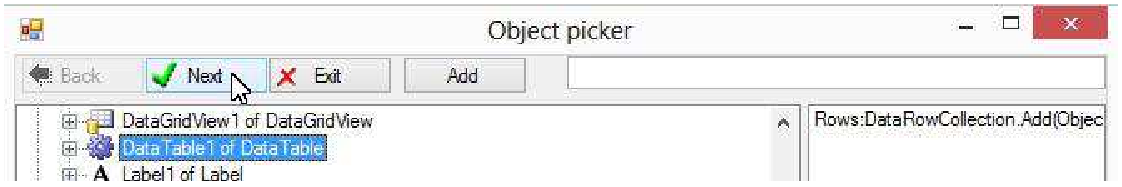
Now DataTable1 contains prices data for all instruments. We may set the DataSource of the DataGridView to DataTable1 so that the DataGridView will display the data.



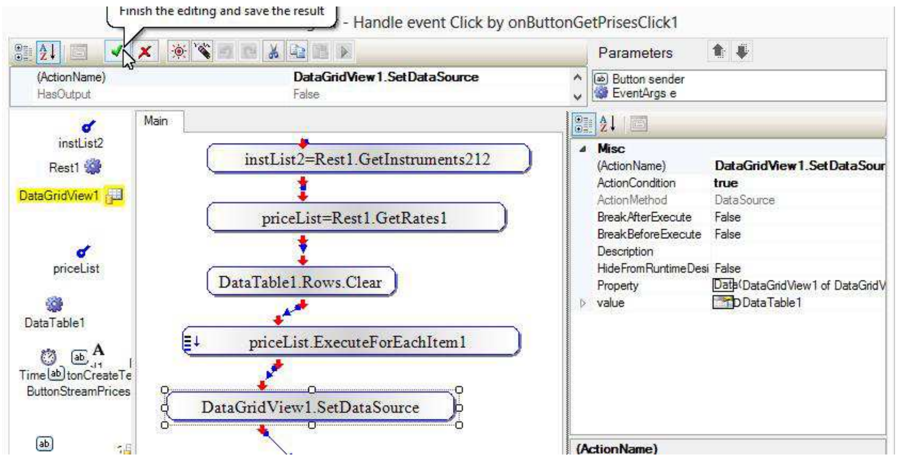


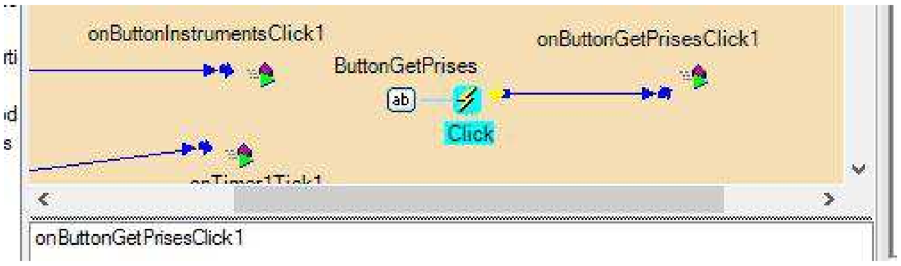
(ActionName)	**DataGridView1.SetDataSource1**
Description	
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	**true**
ActionMethod	DataSource
Property	Data(DataGridView1 of DataGridView.DataSource
value	





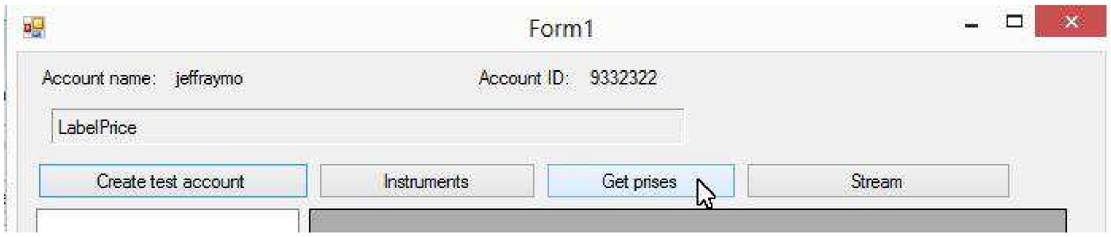
That is all we need for the button event handler:


Finish the editing and save the result
- Handle event Click by onButtonGetPrisesClick1
Parameters
Button sender
EventArgs e
(ActionName) DataGridView1.SetDataSource
HasOutput False
instList2
Rest1
DataGridView1
priceList
DataTable1
Main
instList2=Rest1.GetInstruments212
priceList=Rest1.GetRates1
DataTable1.Rows.Clear
priceList.ExecuteForEachItem1
DataGridView1.SetDataSource
Misc
(ActionName) DataGridView1.SetDataSour
ActionCondition true
ActionMethod DataSource
BreakAfterExecute False
BreakBeforeExecute False
Description
HideFromRuntimeDesi False
Property Data(DataGridView1 of DataGridV
value DataTable1
(ActionName)

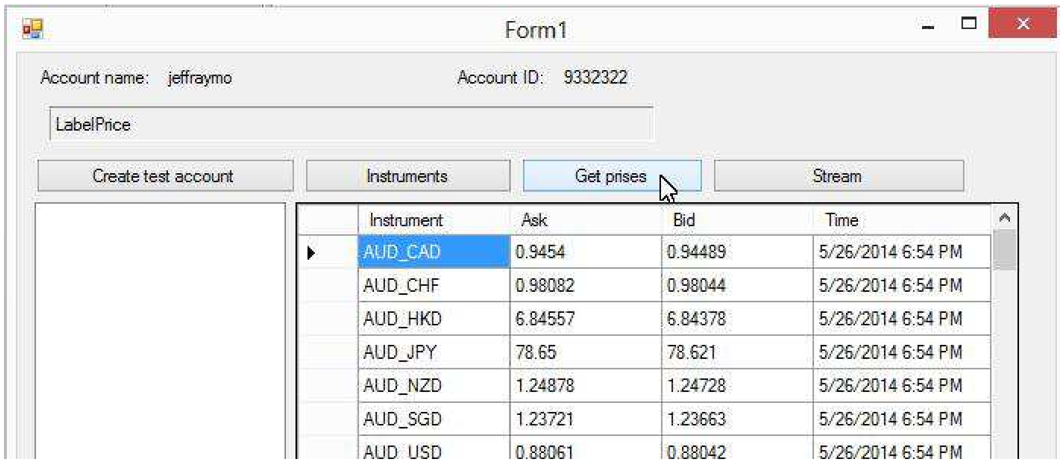


Test

Compile and run the application. The form appears. Click “Create test account”. Click “Get prices”:



After a while, prices appear in the DataGridView:



Instrument	Ask	Bid	Time
AUD_CAD	0.9454	0.94489	5/26/2014 6:54 PM
AUD_CHF	0.98082	0.98044	5/26/2014 6:54 PM
AUD_HKD	6.84557	6.84378	5/26/2014 6:54 PM
AUD_JPY	78.65	78.621	5/26/2014 6:54 PM
AUD_NZD	1.24878	1.24728	5/26/2014 6:54 PM
AUD_SGD	1.23721	1.23663	5/26/2014 6:54 PM
AUD_USD	0.88061	0.88042	5/26/2014 6:54 PM

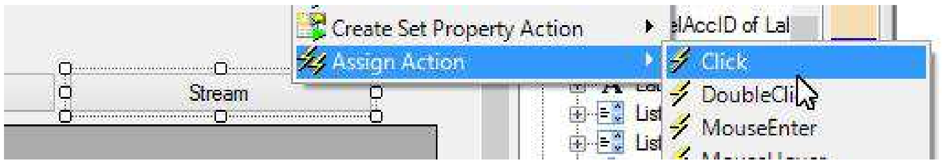
Price Streaming

Method StartGettingRates starts getting price streaming. Once a StartGettingRates action is executed, following events may occur:

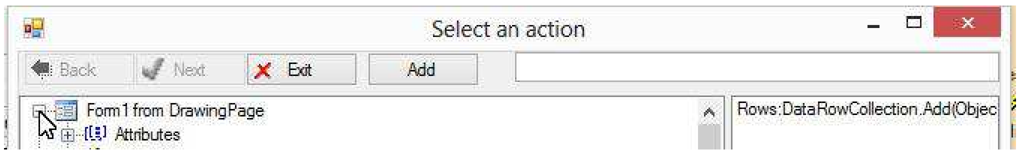
- RateDataArrive – It occurs when price data arrive
- HeartBeat – It occurs when heart beat signal arrive
- StreamError – It occurs when there is an error in processing streaming data.

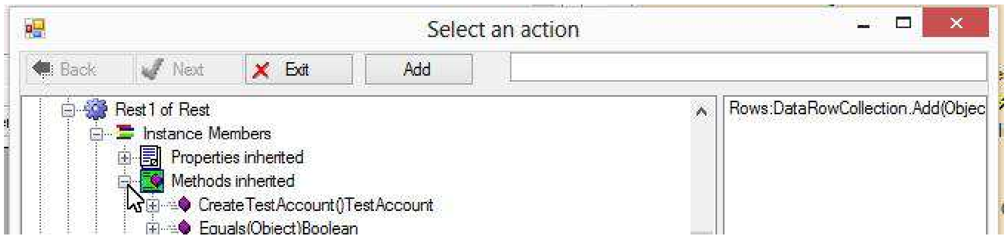
Start price streaming

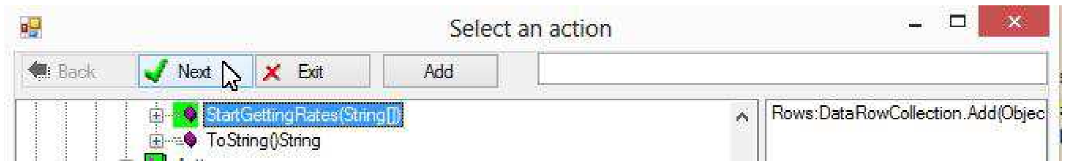
We use a button to start price streaming:



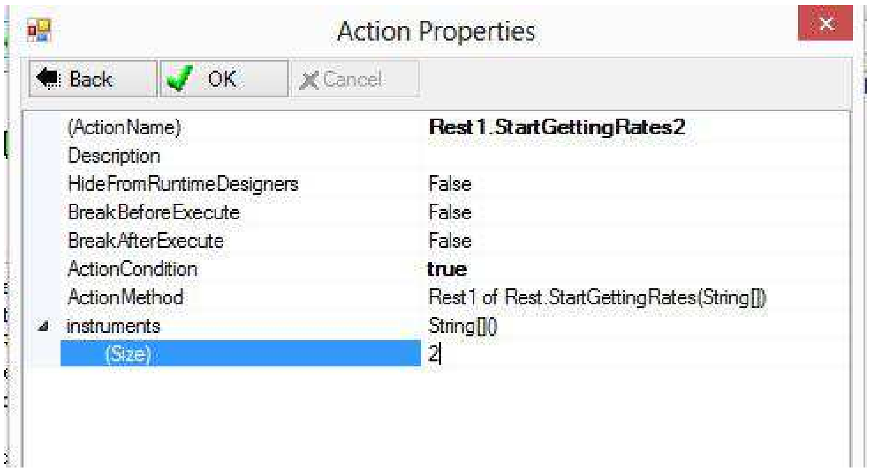
Select StartGettingRates of Rest1:





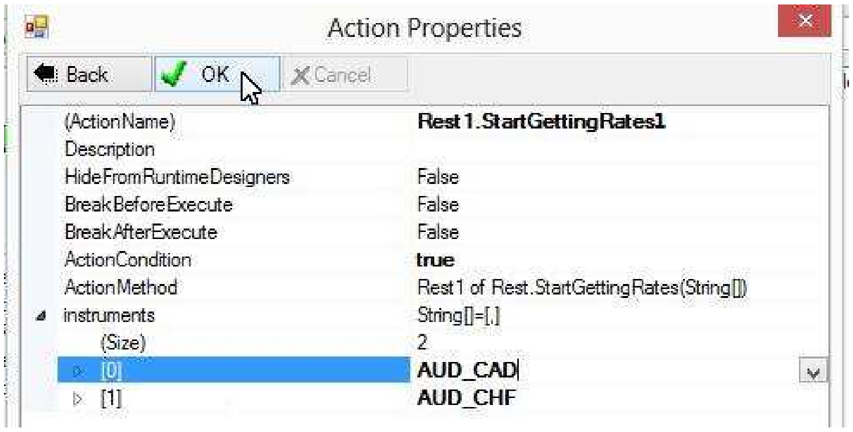


Set “size” to the number of instruments we want to watch. Suppose we want to watch 2 instruments: AUD_CAD and AUD_CHF. Set “size” to 2:



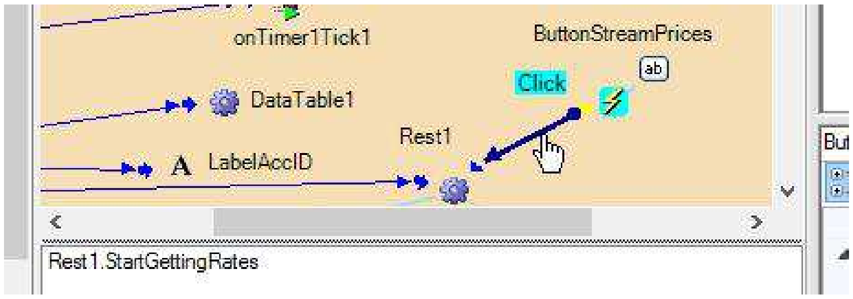
(ActionName)	Rest1.StartGettingRates2
Description	
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Rest1 of Rest.StartGettingRates(String[])
instruments	String[]()
(Size)	2

Set instrument names:



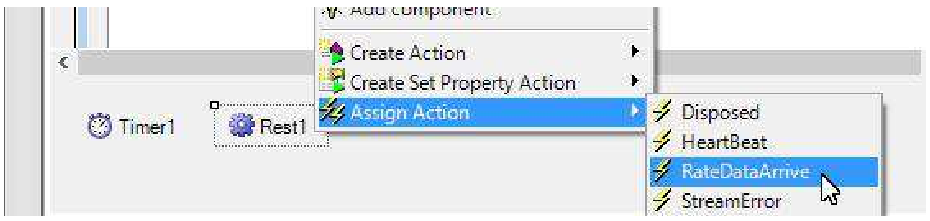
(ActionName)	Rest1.StartGettingRates1
Description	
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Rest1 of Rest.StartGettingRates(String[])
instruments	String[]=[,]
(Size)	2
[0]	AUD_CAD
[1]	AUD_CHF

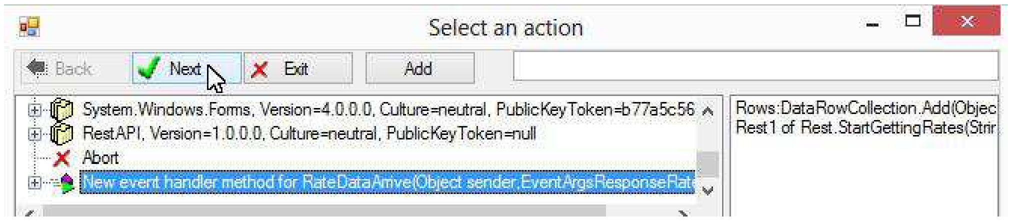
The action is created and assigned to the button:



Handle Event RateDataArrive

Create an event handler to handle RateDataArrive:

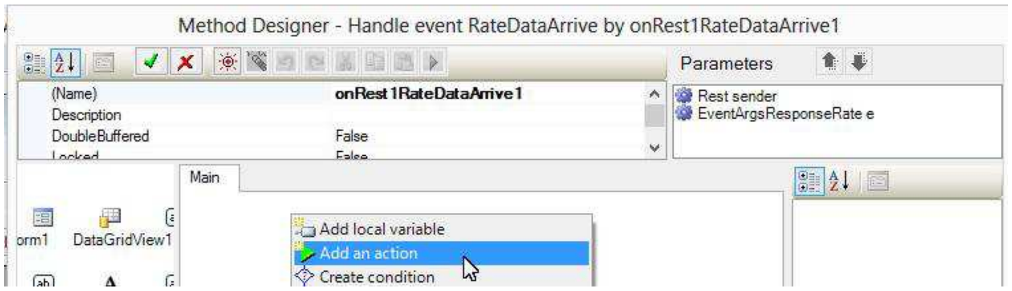




System.Windows.Forms, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c56
RestAPI, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
Abort
New event handler method for RateDataArrive(Object sender,EventArgsResponseRat

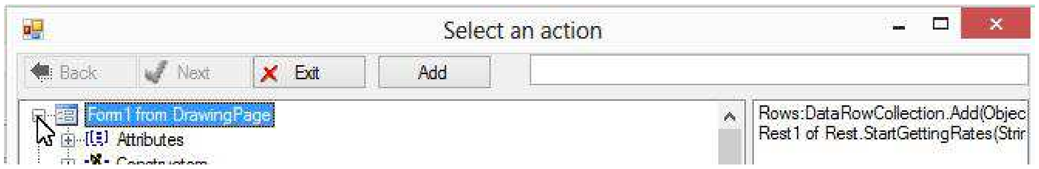
Rows:DataRowCollection.Add(Objec
Rest1 of Rest.StartGettingRates(Strir

For this sample, we simply add the arrived price data to a list box:

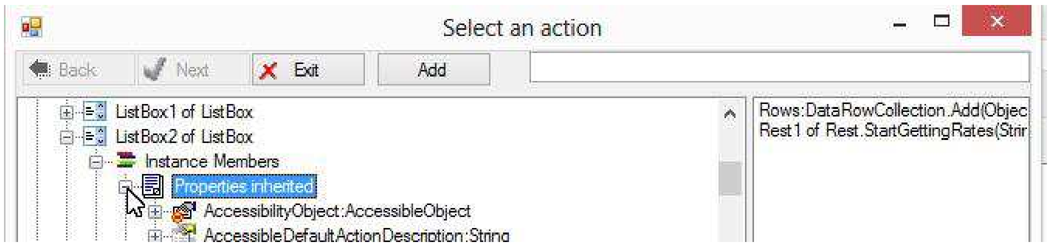


(Name)	onRest1RateDataArrive1
Description	
DoubleBuffered	False

Rest sender
EventArgsResponseRate e

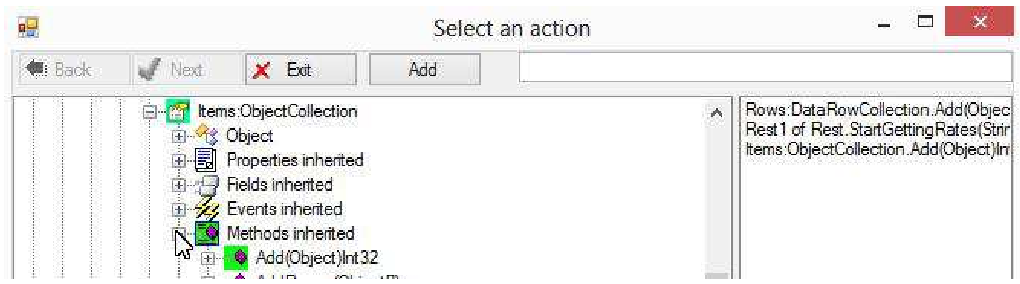


Rows:DataRowCollection.Add(Objec
Rest1 of Rest.StartGettingRates(Strir

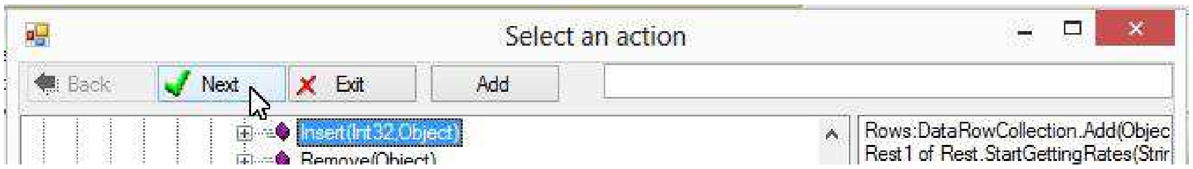


Rows:DataRowCollection.Add(Objec
Rest1 of Rest.StartGettingRates(Strir

Select Insert method:



Rows:DataRowCollection.Add(Objec
Rest1 of Rest.StartGettingRates(Strir
Items:ObjectCollection.Add(Object)In

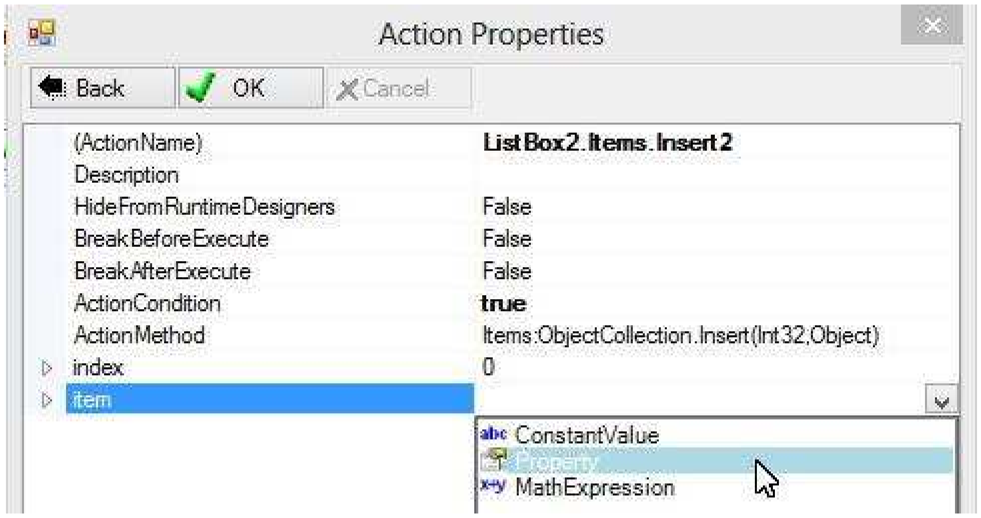


Insert(Int32,Object)
Remove(Object)

Rows:DataRowCollection.Add(Objec
Rest1 of Rest.StartGettingRates(Strir

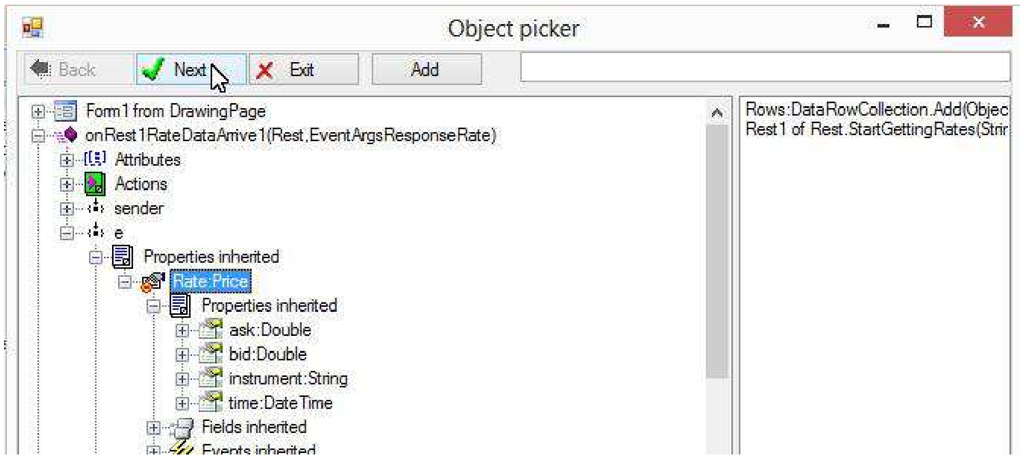
“index” 0 indicates that the new item will be the first list item.

Pass arrived price data to the action:



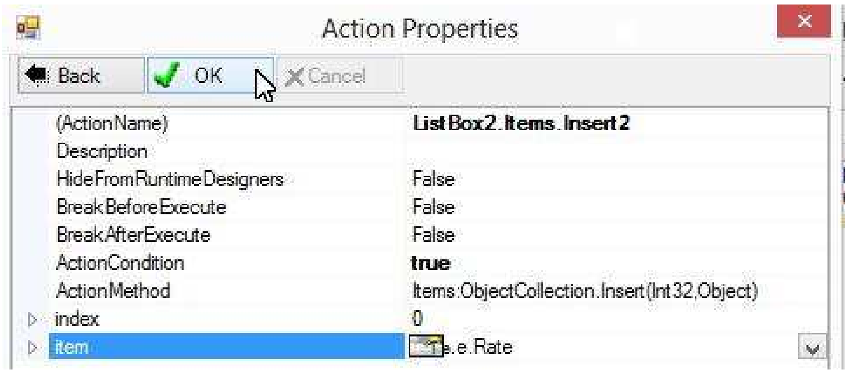
(ActionName)	**ListBox2.Items.Insert2**
Description	
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	**true**
ActionMethod	Items:ObjectCollection.Insert(Int32,Object)
index	0
item	

Rate is the price data arrived. Add it to the list box:



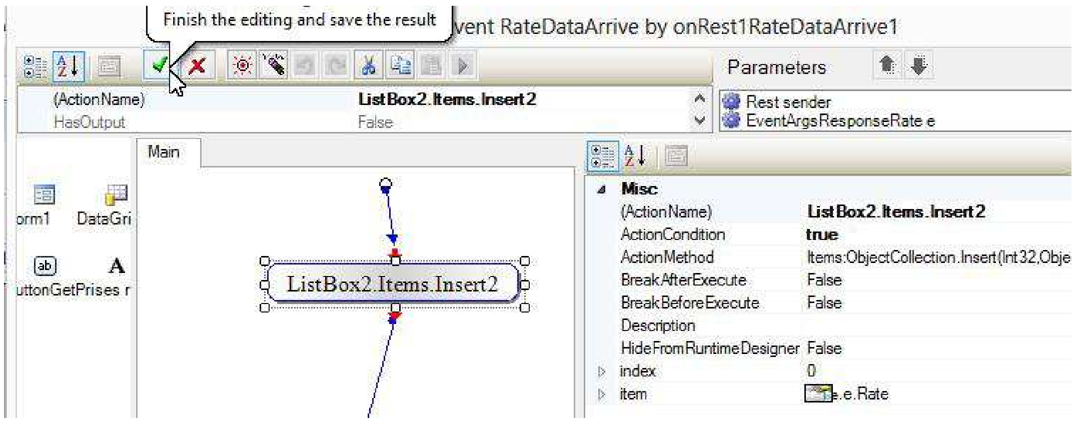
```
Form1 from DrawingPage
onRest1RateDataArrive1(Rest.EventArgsResponseRate)
  Attributes
  Actions
  sender
  e
    Properties inherited
      Rate:Price
        Properties inherited
          ask:Double
          bid:Double
          instrument:String
          time:DateTime
        Fields inherited
        Events inherited
```

Rows:DataRowCollection.Add(Objec
Rest1 of Rest.StartGettingRates(Strir



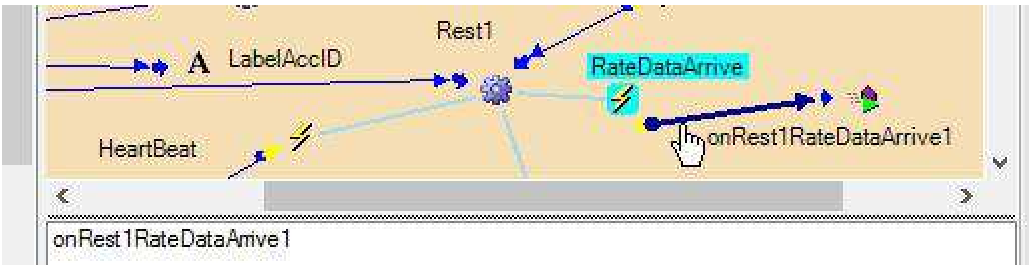
(ActionName)	ListBox2.Items.Insert2
Description	
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Items:ObjectCollection.Insert(Int32,Object)
index	0
item	e.e.Rate

For this sample, that is all for the event handler:



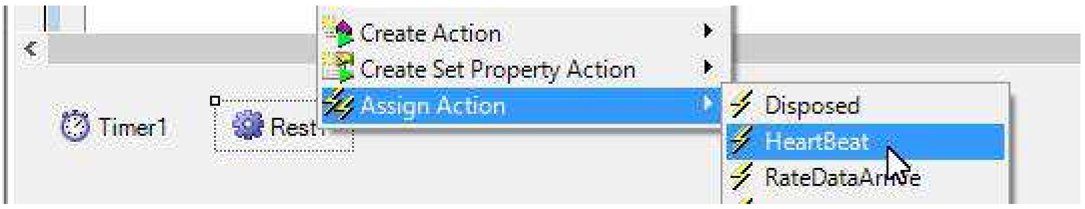
(ActionName)	ListBox2.Items.Insert2
HasOutput	False

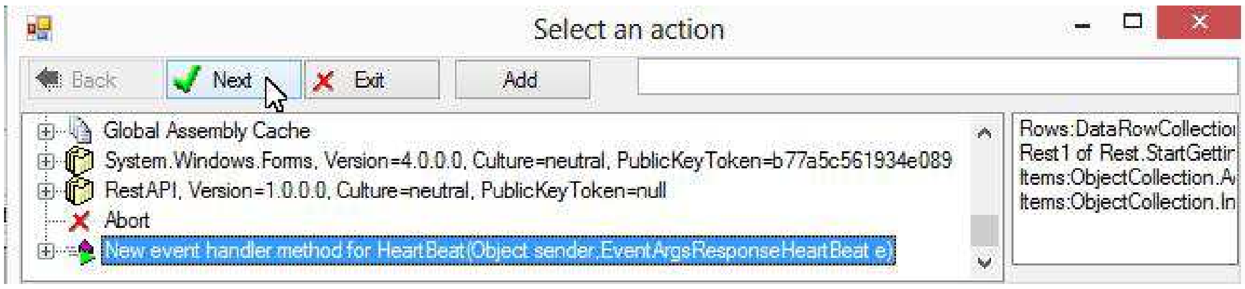
(ActionName)	ListBox2.Items.Insert2
ActionCondition	true
ActionMethod	Items:ObjectCollection.Insert(Int32,Obje
BreakAfterExecute	False
BreakBeforeExecute	False
Description	
HideFromRuntimeDesigner	False
index	0
item	e.e.Rate



Handle Event HeartBeat

Create an event handler to handle HeartBeat:

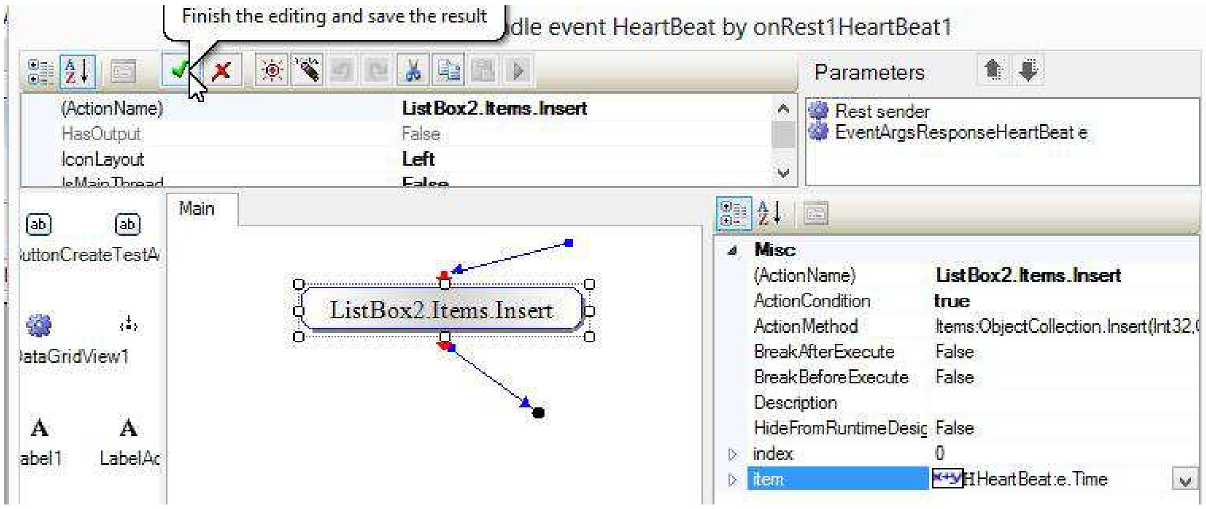




Global Assembly Cache
System.Windows.Forms, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
RestAPI, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
Abort
New event handler method for HeartBeat(Object sender,EventArgsResponseHeartBeat e)

Rows:DataRowCollectio
Rest1 of Rest.StartGettir
Items:ObjectCollection.A
Items:ObjectCollection.In

Add an Insert action of listBox2 to insert heart beat time:

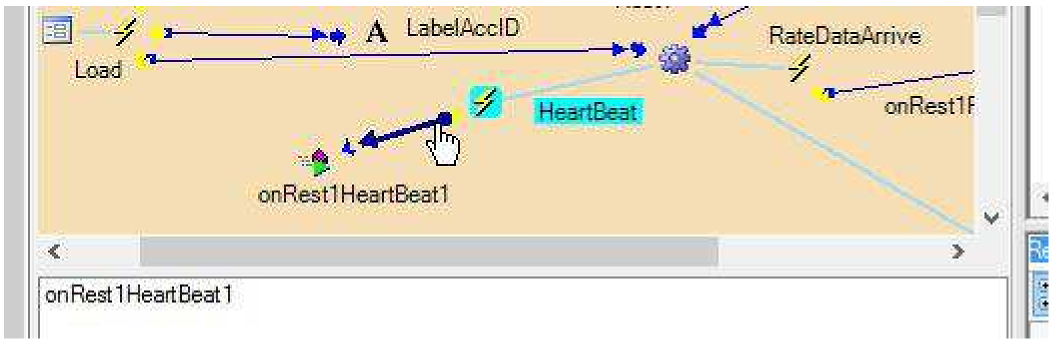

Finish the editing and save the result
dle event HeartBeat by onRest1HeartBeat1
Parameters
Main
ListBox2.Items.Insert
uttonCreateTestA
ataGridView1
abel1
LabelAc

(ActionName)	ListBox2.Items.Insert
HasOutput	False
IconLayout	Left

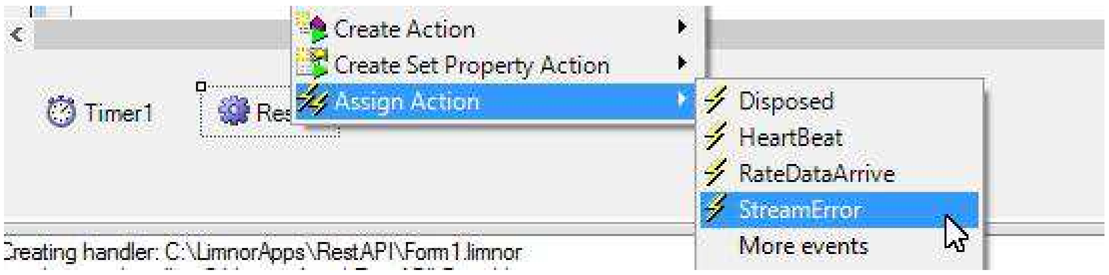
Rest sender
EventArgsResponseHeartBeat e

Misc

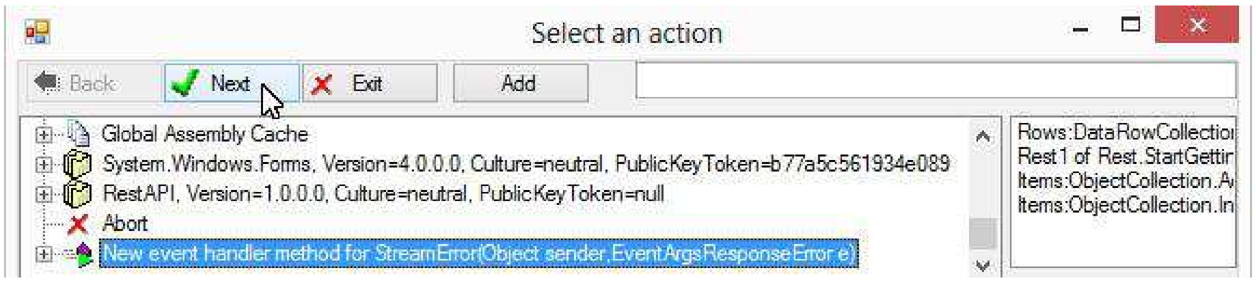
(ActionName)	ListBox2.Items.Insert
ActionCondition	true
ActionMethod	Items:ObjectCollection.Insert(Int32,(
BreakAfterExecute	False
BreakBeforeExecute	False
Description	
HideFromRuntimeDesig	False
index	0
item	HeartBeat:e.Time



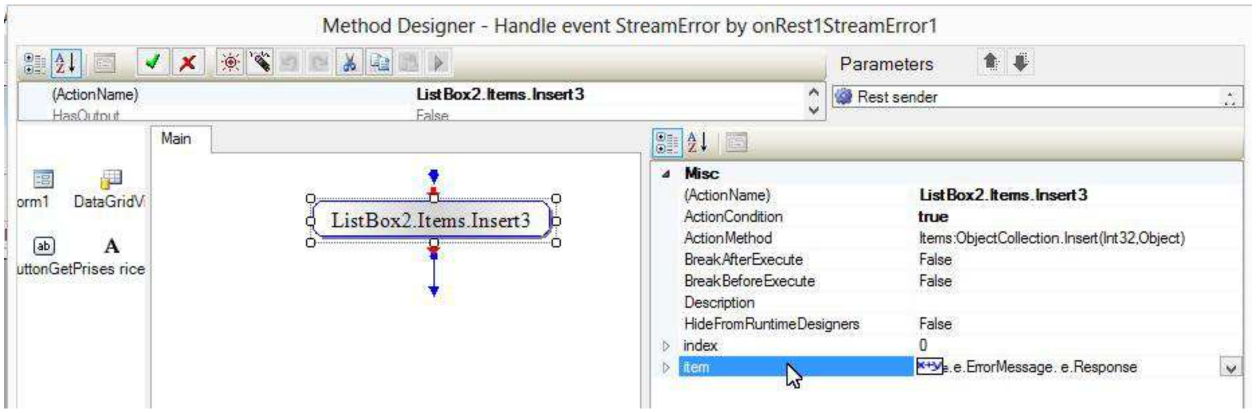
Handle Event StreamError



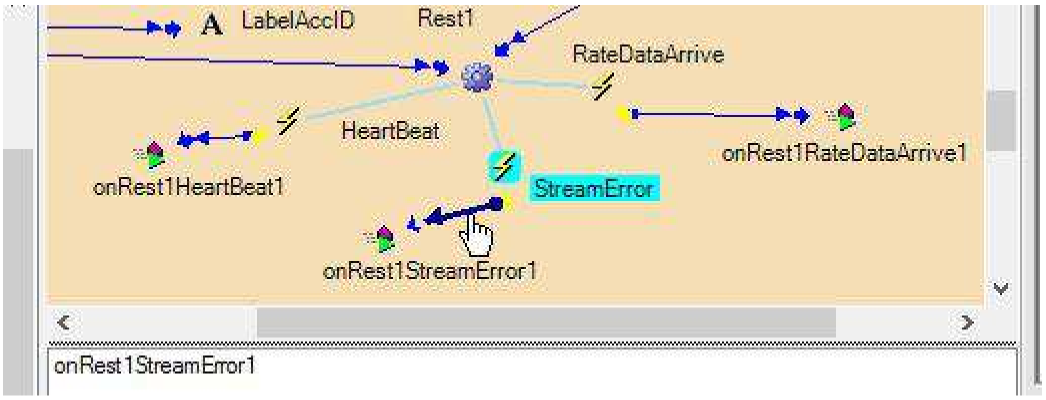
Creating handler: C:\LimnorApps\RestAPI\Form1.limnor


Select an action
Back
Next
Exit
Add
Global Assembly Cache
System.Windows.Forms, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
RestAPI, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
Abort
New event handler method for StreamError(Object sender,EventArgsResponseError e)
Rows:DataRowCollectio
Rest1 of Rest.StartGettir
Items:ObjectCollection.A
Items:ObjectCollection.In

Add an Insert action of listBox2 to show error message and response string:



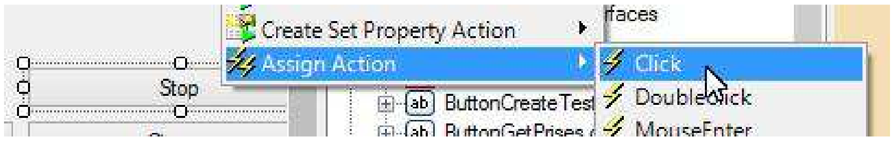
(ActionName)	ListBox2.Items.Insert3
ActionCondition	true
ActionMethod	Items:ObjectCollection.Insert(Int32,Object)
BreakAfterExecute	False
BreakBeforeExecute	False
Description	
HideFromRuntimeDesigners	False
index	0
item	e.ErrorMessage. e.Response



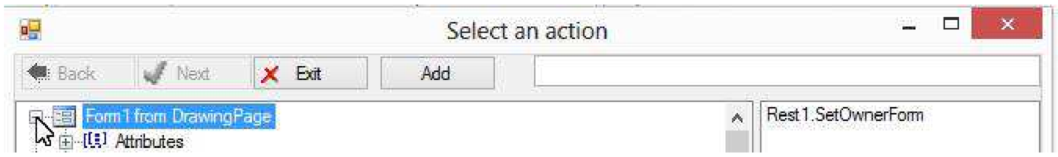
Stop Price Streaming

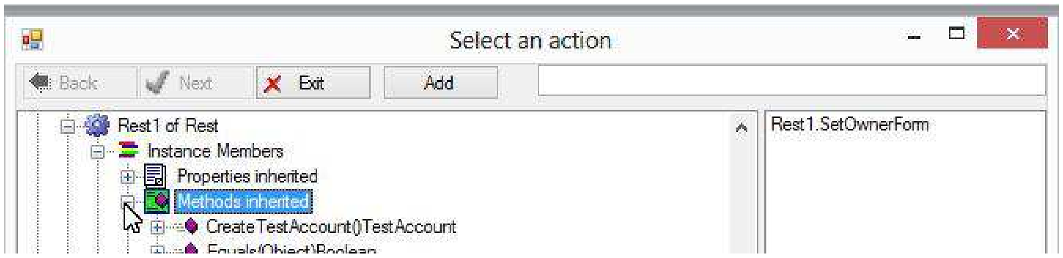
Price streaming keeps a live web connection with the web server. You must disconnect it before your application ends.

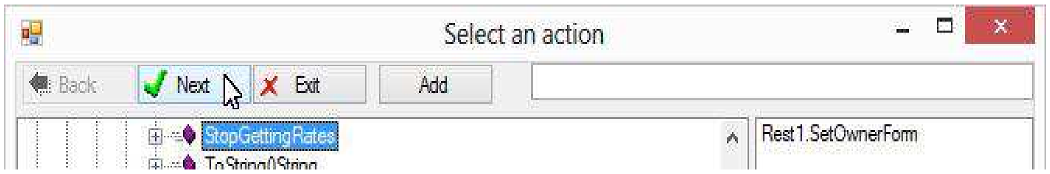
We use a button to stop price streaming:

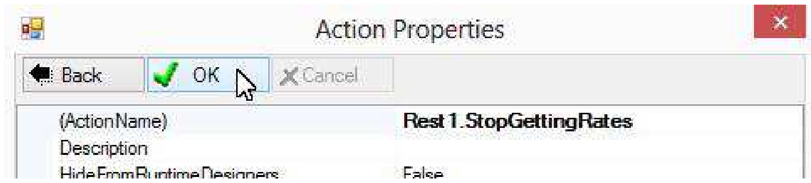


Select StopGettingRates:



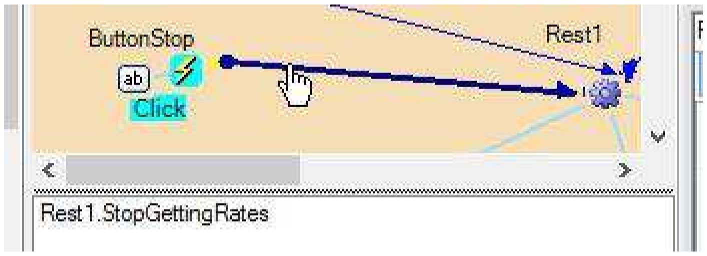




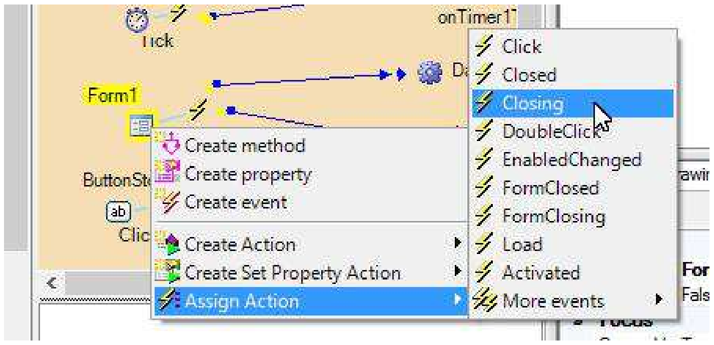


(ActionName)	Rest1.StopGettingRates
Description	
HideFromRuntimeDesigners	False

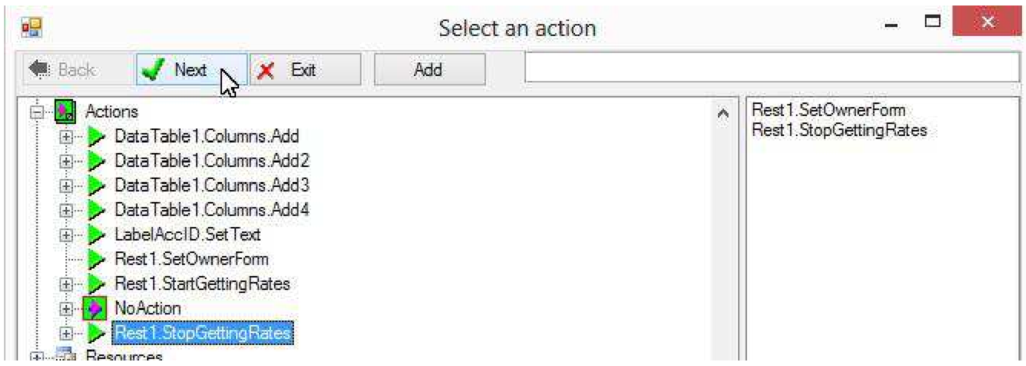
The action is created and assigned to the button:



It is a good idea to assign the action to Closing event of the form so that if the user forgets to click the button before closing the form then the price streaming still will be stopped.



Select the action:

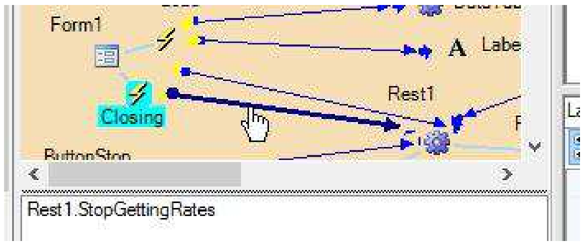


Actions
- DataTable1.Columns.Add
- DataTable1.Columns.Add2
- DataTable1.Columns.Add3
- DataTable1.Columns.Add4
- LabelAccID.SetText
- Rest1.SetOwnerForm
- Rest1.StartGettingRates
- NoAction
- Rest1.StopGettingRates

Resources

Rest1.SetOwnerForm
Rest1.StopGettingRates

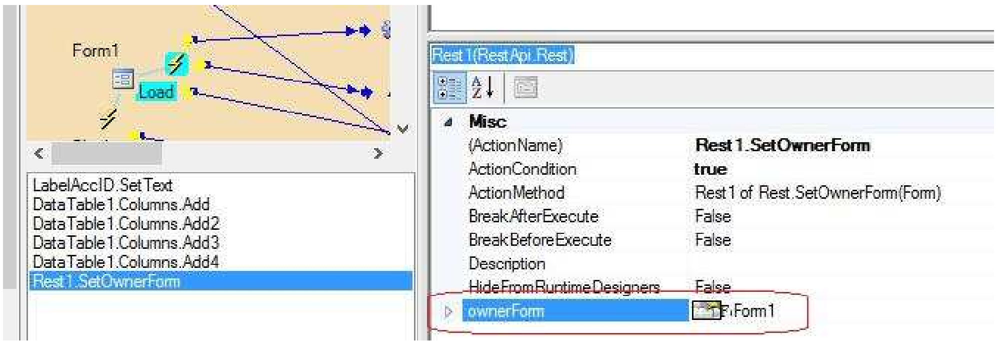
The action is assigned to the Closing event:



Rest1.StopGettingRates

Test

If you are using a Limnor Studio older than 5.6.1.629 then you need to add a SetOwnerForm action at the event of form Load:

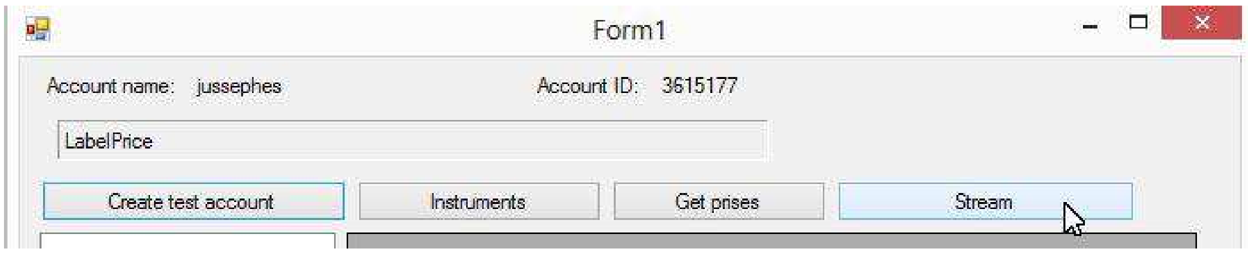


LabelAccID.SetText
DataTable1.Columns.Add
DataTable1.Columns.Add2
DataTable1.Columns.Add3
DataTable1.Columns.Add4
Rest1.SetOwnerForm

(ActionName)	**Rest1.SetOwnerForm**
ActionCondition	**true**
ActionMethod	Rest1 of Rest.SetOwnerForm(Form)
BreakAfterExecute	False
BreakBeforeExecute	False
Description	
HideFromRuntimeDesigners	False
ownerForm	Form1

Note that the “ownerForm” is set to the current form. If you are using a newer version of Limnor Studio then you do not need to create the action.

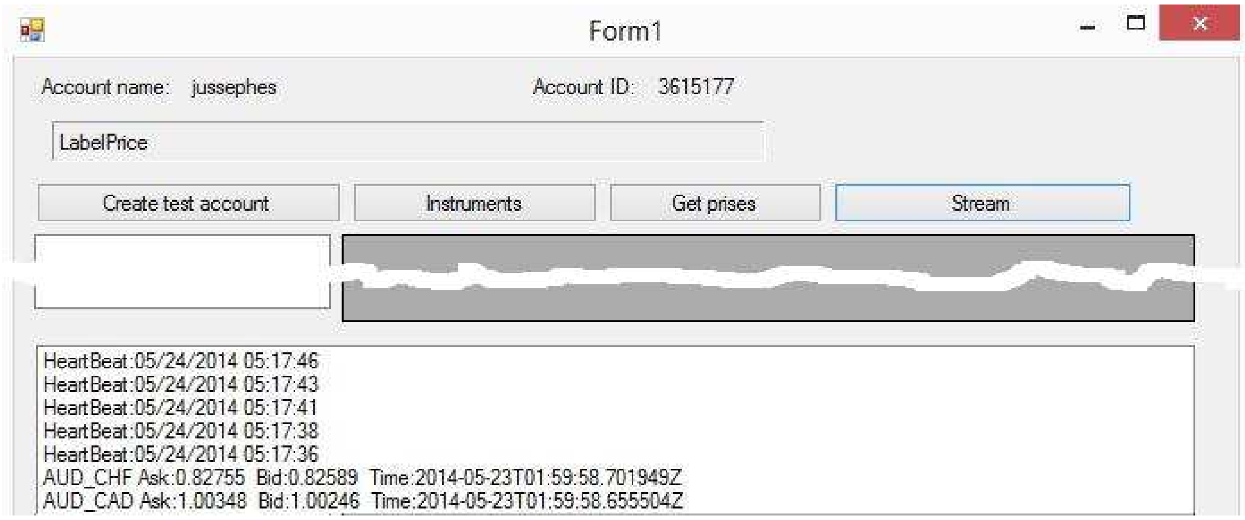
Compile and run the application. Click button “Stream”:



Account name: jussephes Account ID: 3615177

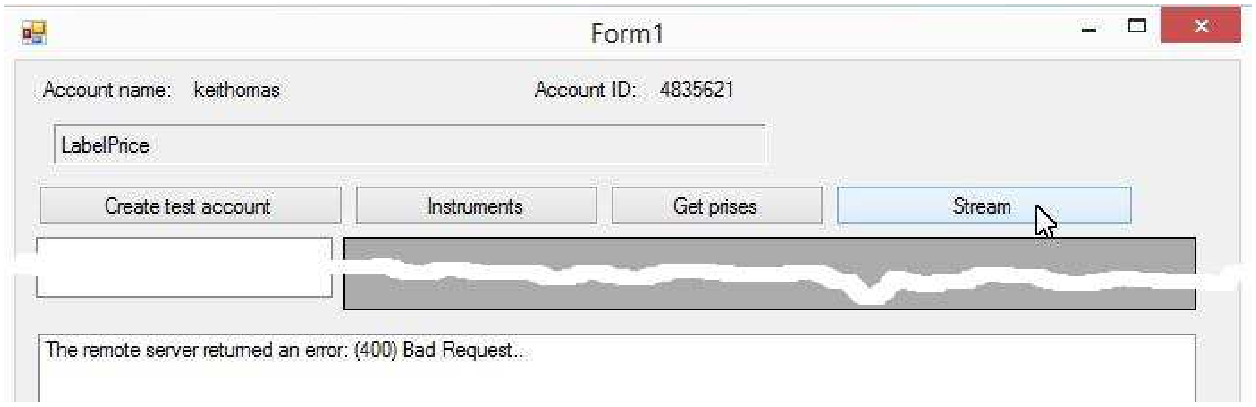
LabelPrice

List box2 shows arrived data:



```
HeartBeat:05/24/2014 05:17:46
HeartBeat:05/24/2014 05:17:43
HeartBeat:05/24/2014 05:17:41
HeartBeat:05/24/2014 05:17:38
HeartBeat:05/24/2014 05:17:36
AUD_CHF Ask:0.82755  Bid:0.82589  Time:2014-05-23T01:59:58.701949Z
AUD_CAD Ask:1.00348  Bid:1.00246  Time:2014-05-23T01:59:58.655504Z
```

If an error occurs then the error message is displayed in the list box:



```
The remote server returned an error: (400) Bad Request..
```

Feedback

Please send your feedback to support@limnor.com