

PROGRAMMING STATISTICA FROM .NET

Virtually every aspect of *STATISTICA* is exposed as a set of COM interfaces that are registered on a machine when *STATISTICA* is installed. Since .NET-based languages cannot communicate with COM directly, a wrapper class called the COM Interop can be utilized to integrate the *STATISTICA* libraries into your .NET project. The COM Interop layer is created automatically by the Visual Studio .NET IDE when you import a COM interface. The COM Interop layer handles all of the details regarding interacting with the COM libraries in .NET. With the COM Interop layer in place, the *STATISTICA* COM interfaces behave like any other .NET object.

Adding the *STATISTICA* 6 Object Library into your .NET project

The .NET Interop layer is created automatically by adding the desired *STATISTICA* COM interfaces into your .NET project. “*STATISTICA* 6 Object Library” is the base *STATISTICA* COM library. To add the *STATISTICA* 6 Object Library to a .NET project, first select the desired .NET project in Solution Explorer, and then select **Add References** from the shortcut menu (accessed by right-clicking on the .NET project). The **Add Reference** dialog will be displayed.

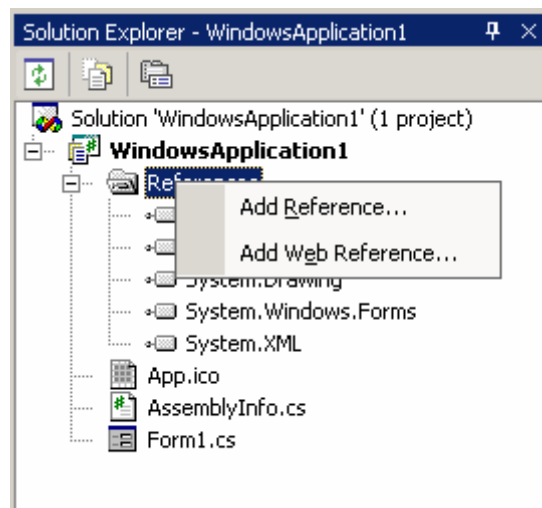


Figure 1 - The References shortcut menu is displayed.

In the **Add Reference** dialog, select the **COM** tab. From the **Component Name** list, select *STATISTICA* 6 Object Library, and click **OK**.

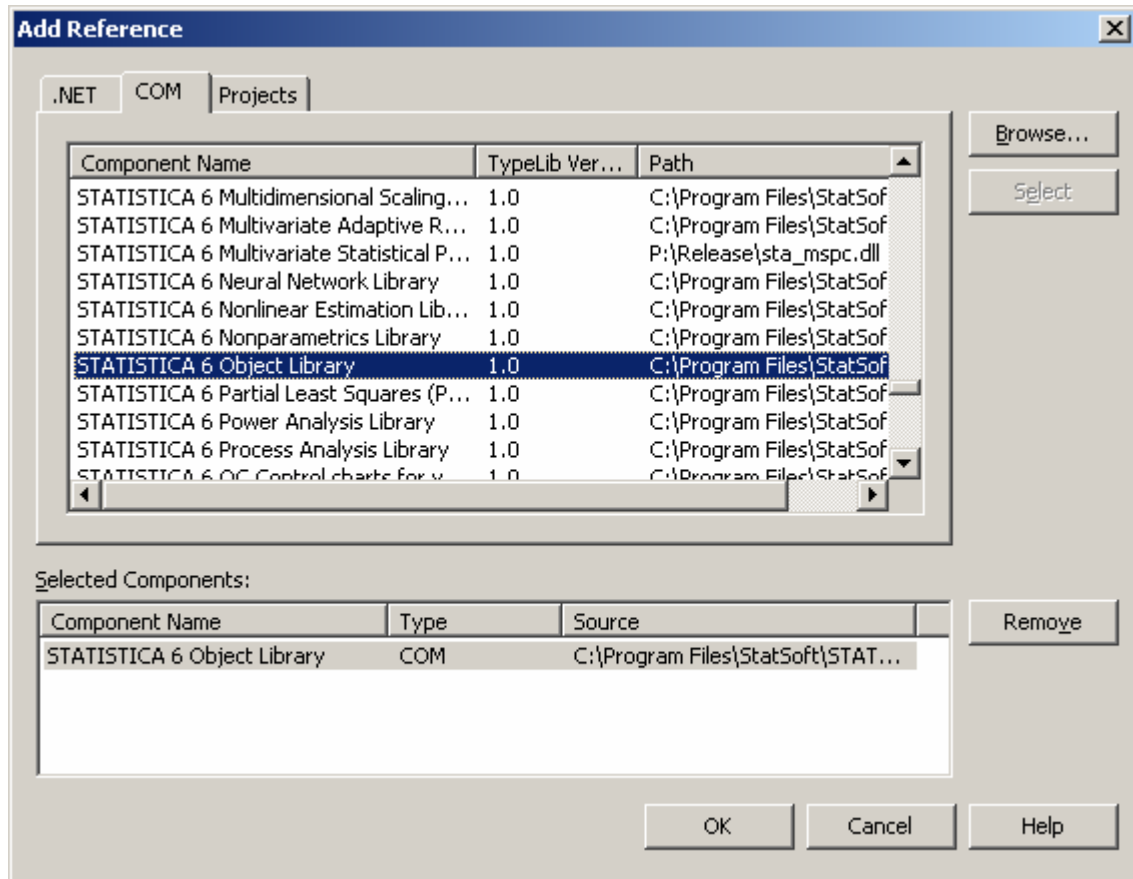


Figure 2 - Adding the *STATISTICA* object library to the project.

At this point, the necessary COM Interop library is created automatically. Under the project *References* node, you will now see the entry *STATISTICA*,

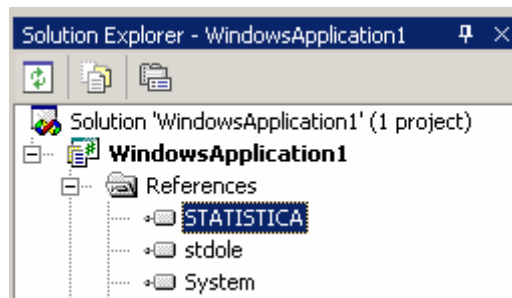


Figure 3 – The *STATISTICA* reference is added to the project.

The file Interop.STATISTICA.dll is also added to the project output directory. The *STATISTICA* COM Interop library is stored in this file. To view the *STATISTICA* object library from your .NET project, right-click on the *STATISTICA* reference, and from the shortcut menu, select **View in Object Browser**.

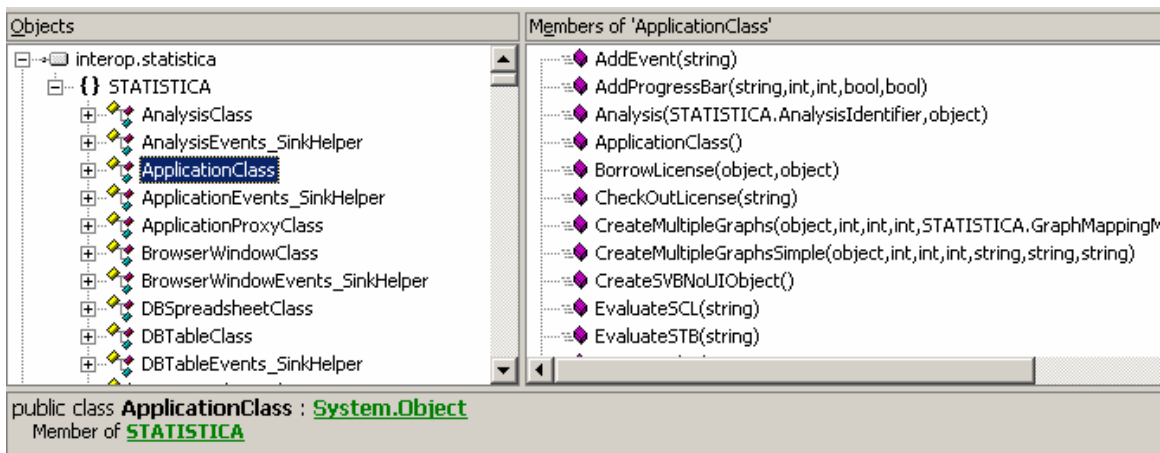


Figure 4 - STATISTICA object library shown in the .NET Object Browser.

Manually Creating the COM Interop Library

It is also possible to create the COM Interop library manually and import it into your .NET project. This gives you the ability to specify a different name for the Interop DLL as well as define a custom namespace. The program that enables you to create an Interop is TLBIMP.EXE. From a Visual Studio command prompt, execute TLBIMP with an initial parameter of the type library source. In the example below, the output DLL name and namespace are also specified:

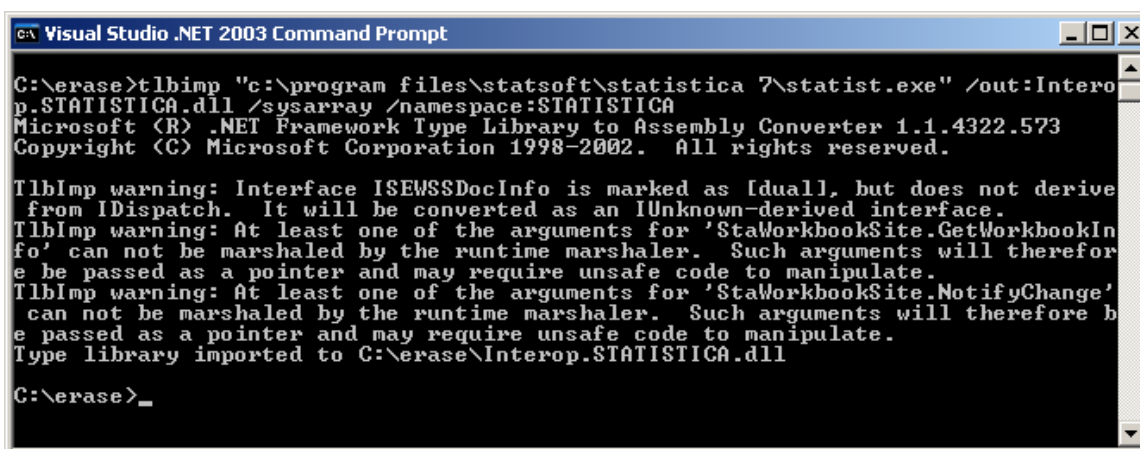


Figure 5 - Using TLBIMP to generate an Interop DLL.

In this example, we reference the file STATIST.EXE since that executable contains the "STATISTICA 6 Object Library" type library. Once the Interop DLL is generated, you can add it to your .NET project by selecting **Add Reference** from the Solution Explorer as before, but this time click the **Browse...** button to select the newly created Interop DLL.

Supporting Multiple Versions of STATISTICA

To support multiple version numbers of STATISTICA, it is necessary to maintain separate STATISTICA 6 Object Library Interop DLL's for each version number of STATISTICA you want to support. You can use the TLBIMP command to generate Interop DLLs against specific versions of STATIST.EXE. When

distributing the application, make sure the correct version of the *STATISTICA* Interop DLL is deployed with your .NET application.

For any other *STATISTICA* libraries, such as *STATISTICAGraphics* or the various analytical modules, it is possible to maintain a single Interop DLL generated from the lowest version number of *STATISTICA* that you need to support. You can use the **TLBIMP.EXE** utility to generate the Interop DLL from the older version number DLL (such as *STA_MGRA.DLL* in the case of *STATISTICAGraphics*). The generated Interop should then be added as a reference to the .NET project (as opposed to the traditional approach of adding a reference to the COM library and letting VS.NET create the Interop DLL automatically).

Instantiating *STATISTICA*

Because of its COM architecture, *STATISTICA* can be incorporated into many different development environments. When using *STATISTICA* from an external development environment, it is necessary to have a top level object called the application object. The application object is the application itself and will contain other objects (for example, spreadsheets and graphs), but access to these other objects is restricted unless the application object is running.

Assuming you are using the default namespace “*STATISTICA*”, the interface you should declare your variable as is *STATISTICA.Application*. To create an instance of *STATISTICA*, set your variable equal to a “*new STATISTICA.ApplicationClass()*”.

```
STATISTICA.Application pApp = (STATISTICA.Application)
    new STATISTICA.ApplicationClass();

pApp.Visible = true;
```

Figure 6 - Creates a *STATISTICA* instance and makes it visible.

When an instance of the *STATISTICA.ApplicationClass* is created, a *STATIST.EXE* process will be launched. This is equivalent to launching *STATISTICA* from the Start menu. The *STATISTICA* instance is initially hidden but can be made visible. Since it is a separate process, all calls to this instance are made out of process.

The Library Version of *STATISTICA*

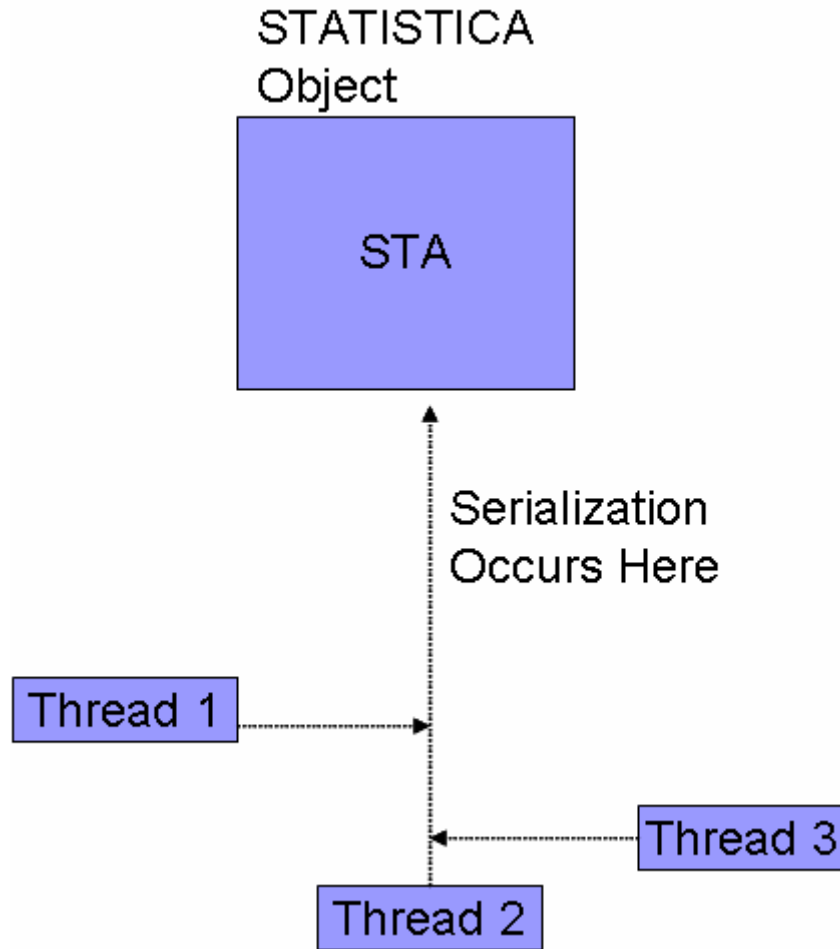
In addition to the *STATISTICA.Application* object, there is also a lighter-weight, higher-performance version of the object called *STATISTICA.Library*. The Library version is licensed separately and therefore may not be available with your installation. It contains identical interfaces as the *STATISTICA.Application* library. Any existing code that uses the Application object can be replaced with the Library object.

The main restriction is that the *STATISTICA* UI features are not available from the Library version. Therefore, in the example above, if the Application object was instantiated as a new *STATISTICA.LibraryClass*, it would not be possible to make the object visible (and show the *STATISTICA* interface).

The Library version of *STATISTICA* is loaded in process, which means accessing its COM interfaces is more efficient than using the Application version of the object (which is loaded out of process). Since it is loaded in process, multiple versions of the library cannot be instantiated. Normally, you would only instantiate one Library object or one Application object in your program.

Threading

Both the Library and Application versions of *STATISTICA* use the single threaded apartment model. Multiple threads accessing an instance of the *STATISTICA* COM interface will queue until previous requests are processed. The architecture of *STATISTICA* is best suited to process requests from one thread at a time.



Optimum performance of the library version occurs when accessed from an STA. Without it, COM calls must marshal across threads. This is immaterial for using the Application object where all calls must be marshaled out of process.

Creating New *STATISTICA* Documents

When creating a *STATISTICA* document such as a new Spreadsheet or Workbook object, you should not instantiate a new instance of that document class. For example, for spreadsheets there is a *STATISTICA.SpreadsheetClass* that you can use from .NET as shown in the example below:

```

STATISTICA.Spreadsheet pSS = (STATISTICA.Spreadsheet)
    new STATISTICA.SpreadsheetClass();

```

Figure 7 – The wrong way to instantiate a STATISTICA document object.

Instead, you should use the corresponding STATISTICA.Application object methods. In the case of spreadsheets, there is an Application.Spreadsheets.New method you can use.

:

```

STATISTICA.Application pApp = (STATISTICA.Application)
    new STATISTICA.ApplicationClass();

STATISTICA.Spreadsheet pSS = (STATISTICA.Spreadsheet)
    pApp.Spreadsheets.New("My new spreadsheet");

```

Figure 8 – The correct way to instantiate a STATISTICA document object.

Failure to use the Application object methods to create *STATISTICA* documents could cause extra *STATISTICA* instances to be created unintentionally.

Setting Properties with Parameters in .NET

C# does not support properties with parameters other than indexers. An example is the Spreadsheet.VariableName property, which is used to get/set variable names in a spreadsheet. In languages such as VB/VB.NET, you can access the properties with parameters directly.

```

Dim spr As New Spreadsheet

spr.VariableName(1) = "New Variable Name" ' Set a variable name
MsgBox spr.VariableName(1) ' Retrieve a variable name

```

Figure 9 - An SVB example of using a property that needs a parameter.

In C# however, you must use the get_ and set_ accessor methods to accomplish this.

```

STATISTICA.Application pApp = (STATISTICA.Application)
    new STATISTICA.ApplicationClass();

STATISTICA.Spreadsheet pSS = (STATISTICA.Spreadsheet)
    pApp.Spreadsheets.New("My new spreadsheet");

pSS.set_VariableName(1, "New variable name");
MessageBox.Show(pSS.get_VariableName(1));

```

Figure 10 - A C# example of using a property that needs a parameter.

No integer Conversion to Boolean in C#

C# does not allow implicit or explicit conversion from int to bool. By convention, most *STATISTICA* analytic routines use "int" parameters as Booleans. This enables adding new states easily. In languages such as SVB, you can set these properties to True/False, but C# will not allow a Boolean value to be assigned to an integer data type.

This can be problematic when copying recorded analysis macros and adding them to the C# code. The recommended way to work around this is to define constant integers that represent True and False.

```
const int True = 1;  
const int False = 0;
```

Default Parameters in C#

Default parameters don't default in C#. When calling a method, you must always specify all parameters. In C# you can pass the parameter `Type.Missing`, which is equivalent to omitting the parameter in languages such as VB/VB.NET.

Garbage Collection in .NET

In languages such as VB 6 and SVB, garbage collection is deterministic. When all references to an object are released, the object is immediately removed from memory. However, in .NET, garbage collection is only performed when necessary as part of a garbage collection process. Setting an object equal to nothing just flags the object for later garbage collection. In situations where you need deterministic releasing of the object, it is possible to force the object's release by using the `ReleaseComObject` method. Calling `ReleaseComObject` decrements a COM object's reference count directly.

```
STATISTICA._Application pApp = (STATISTICA.Application)  
    new STATISTICA.ApplicationClass();  
  
STATISTICA.Workbook pWB = (STATISTICA.Workbook) pApp.Workbooks.New("Temporary Workbook");  
  
while (System.Runtime.InteropServices.Marshal.ReleaseComObject(pWB) > 0);
```

In the example above, the `ReleaseComObject` method is called in a loop that executes until the value returned equals 0. This is because the `ReleaseComObject` method decrements the reference count by one, so if the reference count was greater than 1 to begin with, you need to call the method multiple times in order to release all references.

U.S. Headquarters: StatSoft, Inc. • 2300 E. 14th St. • Tulsa, OK 74104 • USA • (918) 749-1119 • Fax: (918) 749-2217 • info@statsoft.com • www.statsoft.com

Australia: StatSoft Pacific Pty Ltd.
Brazil: StatSoft South America
Bulgaria: StatSoft Bulgaria Ltd.
Czech Rep.: StatSoft Czech Rep. s.r.o.
China: StatSoft China

France: StatSoft France
Germany: StatSoft GmbH
Hungary: StatSoft Hungary Ltd.
India: StatSoft India Pvt. Ltd.
Israel: StatSoft Israel Ltd.

Italy: StatSoft Italia srl
Japan: StatSoft Japan Inc.
Korea: StatSoft Korea
Netherlands: StatSoft Benelux BV
Norway: StatSoft Norway AS

Poland: StatSoft Polska Sp. z o.o.
Portugal: StatSoft Ibérica Lda
Russia: StatSoft Russia
Spain: StatSoft Ibérica Lda

S. Africa: StatSoft S. Africa (Pty) Ltd.
Sweden: StatSoft Scandinavia AB
Taiwan: StatSoft Taiwan
UK: StatSoft Ltd.