StatSoft®

data analysis • data mining • quality control • web-based analytics

Getting Started with *STATISTICA Enterprise* Programming

2300 East 14th Street

Tulsa, OK 74104

Phone: (918) 749-1119

Fax: (918) 749-2217

E-mail: mailto:DeveloperDocumentation@statsoft.com

Web: www.statsoft.com

Table of Contents

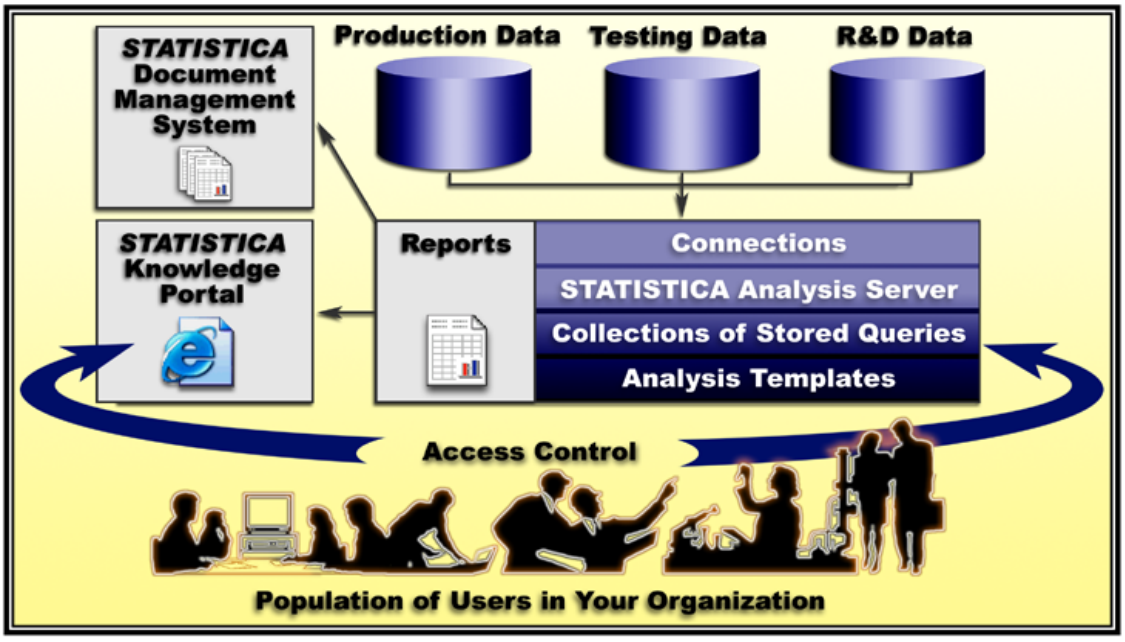
Overview

This is a step-by-step example of *STATISTICA Enterprise* programming, with the ultimate goal of being able to run an existing Analysis Configuration. While each step is broken down into individual segments, each is required and is listed in the order needed.

This document assumes familiarity with general *STATISTICA* programming. *Getting Started with STATISTICA Programming* is a good primer, http://www.statsoft.com/Portals/0/Support/Download/GettingStartedwithSTATISTICAProgramming.pdf.

Introduction to *STATISTICA Enterprise*

STATISTICA Enterprise is an enterprise-wide, multi-user, role-based, secure, thin/thick client-server, administered/guided analytics platform.

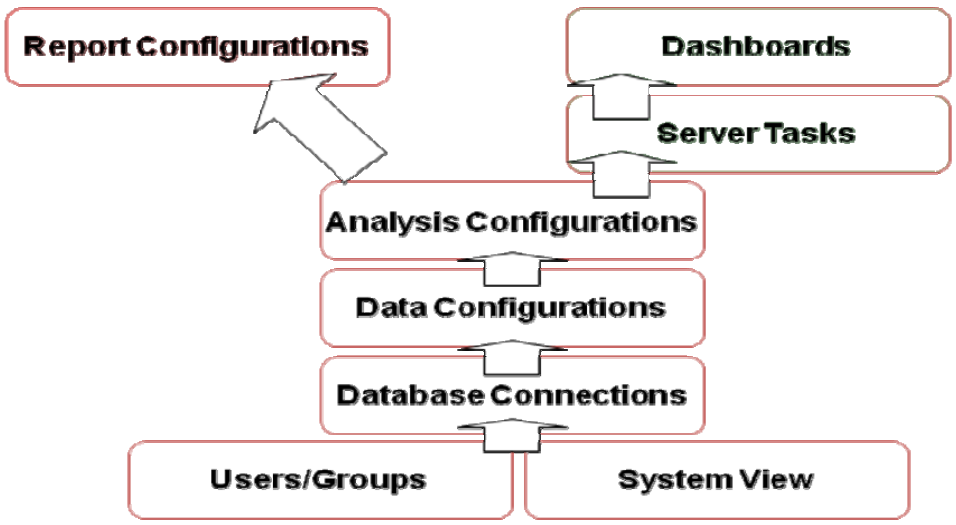


The *STATISTICA Enterprise* platform is built with the latest Microsoft compilers, to run on the Microsoft desktop and server platforms. It:

- Implements all standard interfaces:
  - Built from 14,000+COM functions; all capabilities are (first) available via comprehensive API
  - All data access, data preparation, ETL, graphics, analytics, and data mining functionality is accessible via comprehensive API
  - Can be accessed via .NET (Interop layer)
  - Can be accessed via COM, command line, Excel, etc.
- Database access via:
  - OLEDB, ODBC
  - API Interfaces (e.g., Oracle, for write-back)
- Security is inherited via:
  - NT security (ActiveDirectory)
  - Hooks provided into API to implement other security options, methods

Object Types Overview

STATISTICA Enterprise provides object architecture to manage users, database connections, data configuration, analysis configurations, reports, dashboards, etc. Granular permissions can be created for specific users to provide access only to specific objects in this object hierarchy (specific databases, queries, analysis configurations, reports, etc.)



Accessing and Programming the *STATISTICA Enterprise* Object Model

STATISTICA Enterprise can be accessed through:

- Simple command line interface (e.g., statist.exe /Runmacro=xxx.svb)
- Programming interface (COM interface)
    - Type library
    - From scripting language
    - Compiled programs
        - C++ (adding a reference)
        - .NET (adding a reference, using .NET COM interop)
        - MS PowerShell
- WebServices; calling to a server through SOAP/WebServices interface, using WSDL to describe specific interfaces. (NOTE, this can be used to offload calculations on the server)

Application vs. Library

There are two ways to invoke *STATISTICA*:

- Application
    - *STATIST.EXE* process (out-of-process)
    - Allowed from all versions
    - Cross-process calls can take more time
    - "Process isolation" – *STATISTICA* problems do not affect calling application
- Library
    - *stalib32.dll* process (in-process)
    - In-process call can be faster
    - In-process can affect application

Security

STATISTICA Enterprise integrates with Active Directory security. The security model allows for separate permissions to create data connections and queries, and to use the data returned by the queries. Users do not require permissions to databases; queries run with database admin

credentials. There are separate permissions to specific parts of the system view, specific analysis templates, reports, etc.

Logging on to *STATISTICA Enterprise*

The first step to using *STATISTICA Enterprise* via automation is to log on to the system, just as you would through the interface. On *Enterprise* installations, it is necessary to pass *STATISTICA* your workstation login information when creating a *STATISTICA.Application* object. Calling the function *STATISICA.Application.SWSInfo* immediately after declaring your Application object enables you to do this. *STATISICA.Application.SWSInfo* can accept your user name, password, workstation, and ODBC connection string, although only the first parameter is required. The following example demonstrates how to create a *STATISTICA* instance in Visual Basic on an *Enterprise* system:

```
Sub Main
     Dim x As New STATISTICA.Application
     x.SWSInfo("IsabelleM", "IsabellesPassword", _
          "FloorStation7","DSN=SEWSS;UID=;PWD=;")
End Sub
```

NOTE: If *STATISTICA Enterprise* is not installed, the call to *SWSInfo* will simply be ignored.

Connecting to the Existing Application

To connect to the application object, we will need to create and initialize a *CssSPC* object.

Step 1. Include the necessary references. From within *STATISTICA*, activate the macro to be used. On the ***Edit*** tab in the ***Tools*** group, click ***References***. In the ***References*** dialog, select (check for inclusion) the ***SPC 1.0 Type Library (1.0)*** check box, and click ***OK***.

Step 2. In the active macro, type the following:

```
Dim oSPC As SPCLib.CssSPC
Set oSPC = Application.Parm2Ex(2039)
```

Now, let's examine the code so that we can better understand what is occurring.

```
Dim oSPC As SPCLib.CssSPC
```

Here, we have created an identifier that will be used to reference the *CssSPC* object located in the *SPCLib* library.

```
Set oSPC = oMySTATISTICA.Parm2Ex(2039)
```

In this line, we have initialized the *SPC* object by binding it to the current instance of the existing *STATISTICA* Application. Notice that no login information was included because we are using an existing application object.

Interfacing with *STATISTICA Enterprise*

Now that we have a connection to the application, we need to instantiate a working instance of the object manager, which is the interface to the *Enterprise* repository and *Enterprise* configuration, and associate our instance of *STATISTICA* to the object manger.

Step 1. Include the necessary references. The `ObjectManager` class is located in the *SWLSPCExtension* library, so we will need to include this reference in order to utilize it.

From within *STATISTICA*, activate the macro to be used. On the ***Edit*** tab in the ***Tools*** group, click ***References***. In the ***References*** dialog, select (check for inclusion) the ***STATISTICA SPC Extension 1.0 Object Library (1.0)*** check box.

Step 2. In the active macro, type the following (after the code entered from above):

```
Dim oObjMan As New SWLSpcExtension.ObjectManager
oObjMan.Reconnect(oSPC)
```

Now let's examine the code so that we can better understand what is occurring.

```
Dim oObjMan As New SWLSpcExtension.ObjectManager
```

Here, we have instantiated a working instance of an "Object Manager." The "Object Manager" is, essentially, the interface to the *STATISTICA Enterprise* repository.

```
oObjMan.Reconnect(oSPC)
```

Now we are associating our instance of the *STATISTICA* application to the Object Manager by reconnecting using information contained within the *oSPC* object, which was previously bound to our instance of *STATISTICA*.

Creating a Monitor Service

Keeping in mind that our ultimate goal is to run an existing *Analysis Configuration*, our next step is to create a monitor service using the current session information. To accomplish this, we will need to use some information stored in our *oSPC* object.

Step 1. Include the necessary references. The Monitor Service creator and the monitor service classes are located in the *SWLMonitorService* Library. We will need to include the appropriate reference in order to utilize these classes.

From within *STATISTICA*, activate the macro to be used. On the ***Edit*** tab in the ***Tools*** group, click ***References***. In the ***References*** dialog, select (check for inclusion) the ***STATISTICA SPC Monitor Service 1.0 Object Library (1.0)*** check box.

Step 2. In the active macro, type the following (after the code entered from above):

```
Dim oMonServCreator As New_
    SWLMonitorService.MonitorServiceCreator
```

```
Dim oMonService As SWLMonitorService.MonitorService
Set oMonService = _
          oMonServCreator.CurrentServiceEx(Application)
```

Now let's examine the code so that we can better understand what is occurring.

```
Dim oMonServCreator As New _
       SWLMonitorService.MonitorServiceCreator
```

Here, we are instantiating a working instance of the *MonitorServiceCreator*, which we later use to create a new monitor service.

```
Dim oMonService As SWLMonitorService.MonitorService
```

In this line, we are creating an identifier that will later reference a *MonitorService* object.

```
Set oMonService = _
          oMonServCreator.CurrentServiceEx(Application)
```

Now we are utilizing the *CurrentServiceEx* function to create our monitor service, connecting into the *Enterprise* instance currently attached to the application. If there is not a current *MonitorService* object, this call will create one.

Retrieving a Monitor ID

Note: In the past, *Analysis Configurations* were referred to as *Monitors*, and the object model refers to them as *Monitors* for backward compatibility reasons.

Now that we have monitor service, let's find a specific Analysis Configuration to execute. In order to execute an Analysis Configuration and to perform various other operations, we need to obtain its ID. There are many different ways to determine which Analysis Configuration we want to retrieve an ID for; however, in this example, we will determine the ID by looking for a specific type of *Analysis Configuration* and executing the first *Analysis Configuration* that we encounter. The type of *Analysis Configuration* we will be searching for is an *IQC Monitor*.

Step1. In the active macro, type the following (after the code entered from above):

```
Dim lMonID As Long
For Each Monitor In oObjMan.Monitors
If Monitor.Type = swcIQCMonitor Then
    lMonID = Monitor.ID
    Exit For
End if
Next Monitor
```

If we look at the code line by line, we can closely examine what is occurring

```
For Each Monitor In oObjMan.Monitors
```

Here, we are looping through the monitors (Analysis Configurations) collection located in the object manager.

```
If Monitor.Type = swcIQCMonitor Then
```

If the monitor type is an *IQC monitor*, we will obtain that Analysis Configuration's ID and store it into our *lMonID* variable and then exit the "for" loop.

```
lMonID = Monitor.ID
  Exit For
End if
Next Monitor
```

This continues the loop until we find an IQC monitor.

Obtaining the Run Status of an Analysis Configuration

Now that we have obtained a specific Analysis Configuration, we need to check the run status for our specific Analysis Configuration.

Step 1. In the active macro type the following (after the code entered from above):

```
Dim oMonRunStat As SWLMonitorService.MonitorRunStatus
On Error Resume Next
Set oMonRunStat = oMonService.MonitorRunStatus(lMonID)
On Error GoTo 0
```

Let's examine the code in detail so that we can gain a better understanding of what is occurring.

```
Dim oMonRunStat As SWLMonitorService.MonitorRunStatus
```

Here, we have created an identifier to reference our MonitorRunStatus object.

```
On Error Resume Next
```

The system remembers which Analysis Configurations are currently running; it may be that the Analysis Configuration we have chosen is already executing in the *STATISTICA* application. If this is true, we need to ReRun the monitor rather than execute a new one. In this case, we need to look up to see if there is an existing `MonitorRunStatus` object for this Analysis Configuration. If there is not one, the call will return an error. This is handled by the `On Error Resume Next`; on error, it will just execute the next statement, and our object will be empty.

```
Set oMonRunStat = oMonService.MonitorRunStatus(lMonID)
```

We now obtain the monitor run status object for our specific Analysis Configuration by utilizing the `MonitorRunStatus` function of our `oMonRunStat` object. If the Analysis

Configuration is already present, this will return the current `MonitorRunStatus` object; otherwise, it returns an error that is handled by the `On Error Resume Next/On Error Goto 0` pair. The `MonitorRunStatus` function accepts one parameter, which can be either a monitor ID or a spreadsheet object; for our example we have used the monitor ID we obtained in the proceeding step. The `MonitorRunStatus` function returns a `MonitorRunStatus` object, which we have assigned to our `oMonRunStat` identifier.

```
On Error GoTo 0
```

This returns the error processing back to the default.

Executing a Monitor

Now that we have obtained the run status of our monitor, we are ready to execute the monitor. This can be accomplished with the following:

Step 1. In the active macro, type the following (after the code entered from above):

```
If Not(oMonRunStat Is Nothing) Then
    oMonRunStat.ReRun
Else
    Set oMonRunStat = oMonService.RunMonitor(lMonID,)
End If

OMonRunStatus.WaitForComplete
```

If our `OMonRunStat` is not `Nothing`, we know that a `MonitorRunStatus` already exists for our Analysis Configuration, and we will simply rerun the existing monitor run status.

```
oMonRunStat.ReRun
```

If our `oMonRunStat` is `Nothing`, we know that no monitor run status exists for our Analysis Configuration, so we need to run the Analysis Configuration using `oMonService.RunMonitor`, which returns a monitor run status object, which is assigned to our `oMonRunStat` identifier.

Let's look at the `RunMonitor` function in more detail. The `RunMonitor` function has two parameters. The first parameter is the ID of the Analysis Configuration (Monitor) to be executed; in our example we passed the monitor ID. The second parameter is optional and accepts a `MonitorRunOption`. These represent options that are applied to the monitor when it is executed. These options will be covered in a later section.

```
OMonRunStatus.WaitForComplete
```

The Analysis Configuration is run asynchronously, meaning we can do other things in the SVB code while the Analysis Configuration is running. Since we want to processes the results, we need to wait for the Analysis Configuration to complete. This call will wait until the Analysis Configuration finishes before continuing.

Retrieving Monitor Results

We have now executed our Analysis Configuration, and we want to access the results via automation.

Step 1. In the active macro, type the following (after the code entered from above):

```
Dim oIQCResults As IQCResult
Dim oData As Spreadsheet
Dim oGraph As Graph

Set oIQCResults = oMonRunStat.Result
Set oData = oIQCResults.Data
oData.Visible = True


If (oIQCResults.Outputs.Count > 0) Then
   Set oGraph = oIQCResults.Outputs(1)
   oGraph.Visible = True
End If
```

In this example, we executed an IQC monitor so we know that there is an input data set and at least one output graph. In some instances, more output might be produced and it might be necessary to loop through in order to access all of the outputs.

Let's examine the code line by line. In the following lines, we are creating identifiers to access objects we will need to retrieve the results.

```
Dim oIQCResults As IQCResult
Dim oData As Spreadsheet
Dim oGraph As Graph
```

The results of the Analysis Configuration are stored in an IQC results object that is accessible through the `MonitorRunStatus.Result` property. In the following line, we assign the result to our `IQCResult` object.

```
Set oIQCResults = oMonRunStat.Result
```

Now we can access our input data set using the `Data` property of the `IQCResults` class and assign it to our spreadsheet identifier.

```
Set oData = oIQCResults.Data
oData.Visible = True
```

The IQC outputs created by the IQC analysis are also accessible through the `IQCResults` class in the form of a collection. We access the collection by first checking to see if there are any results, and then using the line `oIQCResults.Outputs(1),` which in our example returns

the first graph produced by our IQC analysis. Note that if we had modified our analysis to produce spreadsheets and graphs as outputs, we would need to check the type of each item so that we could correctly assign it.

Deleting Analysis Configurations That Have Been Run

Now that we are finished with our Analysis Configuration, we want to delete the objects created when we ran it.

Step 1.

```
oMonRunStat.Delete
```

This deletes our current Analysis Configuration objects from the run status. Note that the monitor still exists in *STATISTICA Enterprise*; it is just being deleted from the `MonitorRunStatus`.

Disconnecting the Monitor Service

Now that we are finished, the last thing we want to do is disconnect our monitor service.

Step 1.

```
oMonService.Disconnect
```

This disconnects the monitor service.

Monitor Run Options

Let's look at the options that can be applied to an Analysis Configuration when it is executed, how we apply these options, and what they do.

The `MonitorRunOption` class is located in the `SWLMonitorService` library; we included this reference in one of the steps above to use the monitor service options. To include the reference, do the following:

From within *STATISTICA*, activate the macro to be used. On the ***Edit*** tab in the ***Tools*** group, click ***References***. In the ***References*** dialog, select (check for inclusion) the ***STATISTICA SPC Monitor Service 1.0 Object Library (1.0)*** check box.

Step 1. Create a working object instance of the `MonitorRunOption`.

```
Dim oMonRunOption As New SWLMonitorService.MonitorRunOption
```

Step 2. Specify options using the `MonitorRunOption`.Add method. The add method accepts two parameters. The first parameter is the name of the option to be applied, and the second parameter is the value to be applied for that option. For example:

```
oMonRunOption.Add("RunQuiet", True)
```

Let's examine each of the `MonitorRunOptions` parameters in further detail.

```
oMonRunOption.Add("RunQuiet", boolean)
```

"`RunQuiet`" executes the Analysis Configuration without requiring user intervention. The value for this property is a boolean value. If `True`, the `RunQuiet` option is applied.

```
oMonRunOption.Add("DisableAutoTransfer", boolean)
```

"`DisableAutoTransfer`" disables the automatic updating ability of an Analysis Configuration. The value for this option is a boolean value. If `True`, the `DisableAutoTranser` option is applied.

```
oMonRunOption.Add("CriteriaSourceFile", FilePath)
```

"`CriteriaSourceFile`" allows the Analysis Configuration to use criteria stored in an *.ini* file. The value for the second parameter is a file path to the *.ini* file.

```
oMonRunOption.Add("MaxSample", MaxNumber)
```

"`MaxSample`" sets the maximum number of samples for an IQC Analysis Configuration. The value for the second parameter is the maximum number of samples to be used.

```
oMonRunOption.Add("MaxIQCProject", MaxNumber)
```

"`MaxIQCProject`" sets the maximum number of IQC projects for the Analysis Configuration. The value for the second parameter is the maximum number of IQC projects.

```
MonRunOption.Add("MaxCrosstabColumn", MaxCrosstabColumn)
```

"`MaxCrosstabColumn`" sets the maximum number of crosstab columns that can be created. The value for the second parameter is the maximum number of crosstab columns.

```
MonRunOption.Add("TargetSpreadsheetFileName", FilePath)
```

"`TargetSpreadsheetFileName`" overrides the spreadsheet file name by specifying a new file name or disables the saving of the spreadsheet by specifying "`NoFile`". The value for the second parameter is a string, either the file name to be used or "`NoFile`", to disable the saving of the spreadsheet.

```
MonRunOption.Add("AllowSaveIQCSpec", boolean)
```

"`AllowSaveIQCSpec`" allows the IQC specs to be saved into the *STATISTICA Enterprise* database. The value for the second parameter is a boolean value. If true is used, the specs will be saved in the *STATISTICA Enterprise* database; if False, they will not be saved.

```
MonRunOption.Add("DataOnly", boolean)
```

"`DataOnly`" extracts only the data from the Analysis Configuration. No IQC projects or SVB

programs are executed. The value for the second parameter is a boolean value; if True is used, only the data will be extracted when the Analysis Configuration is executed.

```
MonRunOption.Add("SetName", SetName)
```

"SetName" allows a set name to be specified. The second parameter is a string value representing the set name to be used.

```
MonRunOption.Add("RunSVBVisibly", boolean)
```

"RunSVBVisibly" allows the user to specify whether SVB programs are visible when executed. The second parameter is a boolean value. If True is specified, the SVB will be executed in visible mode. If False, the SVB will not be visible.

```
MonRunOption.Add("Force_Display_SQLCriteria_Dialog_If_Requi
red_Fields_Need_Values", boolean)
```

"Force_Display_SQLCriteria_Dialog_If_Required_Fields_Need_Values" - if a required field needs a value (filtering), the ***SQL Criteria*** dialog will be displayed. The second parameter for this option is a boolean value.

```
MonRunOption.Add("ForceRunNewDataMacro", boolean)
```

"ForceRunNewDataMacro" forces the SVB code associated with automatic updates to be executed, even if automatic updating is disabled. The second parameter for this option is a boolean value.

```
MonRunOption.Add("Debug", boolean)
```

"Debug" allows information to be written to the log file. The second parameter for this option is a boolean value.