Platform SDK Developer's Guide

Stat Server

12/13/2025

Stat Server

Stat Server tracks information about customer interaction networks (contact center, enterprise-wide, or multi-enterprise telephony and computer networks). It also converts the data accumulated for directory numbers (DNs), agents, agent groups, and non-telephony-specific object types, such as email and chat sessions, into statistically useful information, and passes these calculations to other software applications that request data. For example, Stat Server sends data to Universal Routing Server (URS), because Stat Server reports on agent availability. You can also use Stat Server's numerical statistical values as routing criteria.

Stat Server provides contact center managers with a wide range of information, allowing organizations to maximize the efficiency and flexibility of customer interaction networks. For more information about Stat Server, consult the Reporting Technical Reference 8.0 Overview and the Stat Server 8.5 User's Guide.

You can use the Platform SDK to write Java or .NET applications that gather statistical information from Stat Server. These applications may be fairly simple or quite advanced. This article shows how to implement the basic functions you will need to write a simple Statistics application.

A Typical Statistics Application

There are many ways in which you might need to use data from Stat Server, but in most cases, you will use three types of requests:

- `RequestOpenStatistic` and `RequestOpenStatisticEx` are used to ask Stat Server to start sending statistical information to your application. `RequestOpenStatistic` allows you to request information about a statistic that has already been defined in the Genesys Configuration Layer, while you can use `RequestOpenStatisticEx` to define your own statistics dynamically.
- You can use `RequestPeekStatistic` to get the value of a statistic that has already been opened using either `RequestOpenStatistic` or `RequestOpenStatisticEx`. Since it can take a while for certain types of statistical information to be sent to your application, this can be useful if you are writing an application—such as a wallboard application, for instance—for which you would like statistical values to be displayed immediately.
- Use `RequestCloseStatistic` to tell Stat Server that you no longer need information about a particular statistic.

> ### Tip
> When you use `RequestOpenStatistic` and `RequestOpenStatisticEx`, you have to specify a `ReferenceId`, which is a unique integer that allows Stat Server and your application to distinguish between different sets of statistical information. You must also enter this integer in the `StatisticId` field for any request that refers to the statistics generated on the basis of your Open request. For example, if you sent a request for "TotalNumberInboundCalls" for agent 001, you might give the `RequestOpenStatistic` a `ReferenceId` of `333001`. A similar request for agent 002

> might have a `ReferenceId` of `333002`. When you want to peek at the value of "TotalNumberInboundCalls" for agent 001, or close the statistic (or suspend or resume reporting on the statistic), you need to specify a `StatisticId` of `333001` for each of these requests.

Java

Connecting to Stat Server

As mentioned in the article on the architecture, the Platform SDKs uses a message-based architecture to connect to Genesys servers. This section describes how to connect to Stat Server, based on the material in the article on Connecting to a Server.

After you have set up your import statements, the first thing you need to do is create a `StatServerProtocol` object:

```
[Java]

StatServerProtocol statServerProtocol =
  new StatServerProtocol(
    new Endpoint(
      statServerEPName,
      host,
      port));
statServerProtocol.setClientName(clientName);
```

You can also configure your ADDP and warm standby settings at this point, following the example shown in the Connecting to a Server article.

Once your configuration is complete, open the connection to Stat Server:

```
[Java]

try {
        statServerProtocol.open();
} catch (InterruptedException e) {
        e.printStackTrace();
} catch (ProtocolException e) {
        e.printStackTrace();
}
```

Working with Statistics

The Stat Server application object in the Genesys Configuration Layer comes with many predefined statistics. You can also define your own statistics using the options tab of this application object. The Platform SDK allows you to get information about any of these statistics by using

RequestOpenStatistic. There may be times, however, when you want your application to be able to create new types of statistics dynamically. The Platform SDK also supports this, with the use of RequestOpenStatisticEx.

This section will show you how to use RequestOpenStatistic to get information on a predefined statistic. After that, we will give an example of how to use RequestOpenStatisticEx.

The first thing you need to do to use RequestOpenStatistic is to create the request:

```
[Java]

RequestOpenStatistic requestOpenStatistic
        = RequestOpenStatistic.create();
```

Now you need to describe the *statistics object*, that is, the object you are monitoring. This description consists of the object's Configuration Layer ID and object type, and the tenant ID and password:

```
[Java]

StatisticObject object = StatisticObject.create();
object.setObjectId("Analyst001");
object.setObjectType(StatisticObjectType.Agent);
object.setTenantName("Resources");
object.setTenantPassword("");
```

Next, you will specify the StatisticType property, which must correspond to the name of the statistic definition that appears in the options tab. In this case, we are asking for the total login time for an agent identified as "Analyst001":

```
[Java]

StatisticMetric metric = StatisticMetric.create();
metric.setStatisticType("TotalLoginTime");
```

Now you can specify the desired Notification settings. The Statistics Platform SDK supports four ways of gathering statistics:

1. NoNotification allows you to retrieve statistics when you want them.
2. Periodical means Stat Server reports on statistics based on the time period you request.
3. Immediate means Stat Server reports on statistics whenever a statistical value changes. For time-related statistics, Immediate means that Stat Server will report the current value whenever a statistical value changes, but it will also report that value periodically, using the specified notification frequency.
4. Reset means Stat Server reports the current value of a statistic right before setting the statistical value to zero (0).

In this case, we are interested in receiving statistics on a regular basis, so we have asked for a notification mode of Periodical, with updates every 5 seconds, using a GrowingWindow statistic interval. For more information on notification modes, see the section on Notification Modes in the Stat Server 8.5 User's Guide. For more information on statistic intervals, see the section on TimeProfiles in the same guide.

```
[Java]

Notification notification = Notification.create();
notification.setMode(NotificationMode.Periodical);
notification.setFrequency(5);
```

At this point, you can add the information about the statistic object and your notification settings to the request:

```
[Java]

requestOpenStatistic.setStatisticObject(object);
requestOpenStatistic.setStatisticMetric(metric);
requestOpenStatistic.setNotification(notification);
```

Before sending this request, you have to assign it an integer that uniquely identifies it, so that Stat Server and your application can easily distinguish it from other sets of statistical information. Note that you will also need to enter this integer in the `StatisticId` field for any subsequent requests that refer to the statistics generated on the basis of the Open request.

> ### Tip
>
> `ReferenceId` is a unique integer that is specified for identification of requested statistics. If no value is set then this property is assigned automatically just before sending a message; however, if the property has already been assigned then it will not be modified. If you specify this property on your own, you should guarantee its uniqueness. StatServer's behavior was corrected (starting from releases 8.1.000.44 and 8.1.200.14 in corresponding families) so that if two requests are sent with the same `ReferenceId` then an `EventError` message is returned for the second request.

```
[Java]

requestOpenStatistic.setReferenceId(2);
```

Now you can send the request:

```
[Java]

System.out.println("Sending:\n" + requestOpenStatistic);
statServerProtocol.send(requestOpenStatistic);
```

After Stat Server sends the `EventStatisticOpened` in response to this request, it will start sending `EventInfo` messages every 5 seconds. You need to set up an event handler to receive these messages, as discussed in the the Event Handling article.

This is what one such message might look like:

```
'EventInfo' ('2')
message attributes:
REQ_ID [int]    = 4
USER_REQ_ID [int] = -1
TM_SERVER [int] = 1244412448
TM_LENGTH [int] = 0
LONG_VALUE [int] = 0
VOID_VALUE [object] = AgentStatus {
        AgentId = Analyst001
        AgentStatus = 23
        Time = 1240840034
        PlaceStatus =   PlaceStatus = 23
        Time = 1240840034
        LoginId = LoggedOut
```

```
}
```

Creating Dynamic Statistics

As mentioned above, there may be times when you want to get statistical information that has not already been defined in the Configuration Layer. In cases like that, you can use RequestOpenStatisticEx. Before you do, however, you should make sure you understand several topics covered in the Reporting Technical Reference 8.0 Overview and the Stat Server 8.5 User's Guide, including the use of masks.

The first things you need to do in order to use RequestOpenStatisticEx are similar to what we did in the previous section. You will start by creating the request and specifying the statistic object and notification mode, which you will add to the request:

```
[Java]

RequestOpenStatisticEx request =
        RequestOpenStatisticEx.create();

StatisticObject object = StatisticObject.create();
object.setObjectId("Analyst001");
object.setObjectType(StatisticObjectType.Agent);
object.setTenantName("Resources");
object.setTenantPassword("");

Notification notification = Notification.create();
notification.setMode(NotificationMode.Immediate);

request.setNotification(notification);
request.setStatisticObject(object);
```

Now, instead of requesting a pre-defined statistic type, you need to set up your own masks, as described in the section on "Metrics: Their Composition and Definition" in the Reporting Technical Reference 8.0 Overview. The following mask and statistic metric settings give the Current State for the agent mentioned above:

```
[Java]

DnActionMask mainMask = ActionsMask.createDNActionsMask();
mainMask.setBit(DnActions.WaitForNextCall);
mainMask.setBit(DnActions.CallDialing);
mainMask.setBit(DnActions.CallRinging);
mainMask.setBit(DnActions.NotReadyForNextCall);
mainMask.setBit(DnActions.CallOnHold);
mainMask.setBit(DnActions.CallUnknown);
mainMask.setBit(DnActions.CallConsult);
mainMask.setBit(DnActions.CallInternal);
mainMask.setBit(DnActions.CallOutbound);
mainMask.setBit(DnActions.CallInbound);
mainMask.setBit(DnActions.LoggedOut);

DnActionMask relMask = ActionsMask.createDNActionsMask();

StatisticMetricEx metric = StatisticMetricEx.create();
metric.setCategory(StatisticCategory.CurrentState);
metric.setMainMask(mainMask);
metric.setRelativeMask(relMask);
```

```
metric.setSubject(StatisticSubject.DNStatus);

request.setStatisticMetricEx(metric);
```

Once you have set up the masks and the statistic metric, you can create a `ReferenceId` and send the request:

```
[Java]

request.setReferenceId(anIntThatYouSpecify);

System.out.println("Sending:\n" + request);
Message response = statServerProtocol.request(request);
System.out.println("Received:\n" + response);
```

Current Target State Events

You can use `RequestGetStatisticEx` and `RequestOpenStatisticEx` to set up the same type of current target state definitions that Universal Routing Server (URS) uses. (You can also set these up using Configuration Manager.) When this type of request has been sent, Stat Server sends some additional event types:

- EventCurrentTargetStateSnapshot
- EventCurrentTargetStateTargetUpdated
- EventCurrentTargetStateTargetAdded
- EventCurrentTargetStateTargetRemoved

The `Snapshot` event is returned in response to the open, while the `Updated` event is sent as state changes occur. In a situation where you open a CurrentTargetState-based statistic against an agent group, the `Added` and `Removed` messages occur when an agent is added to or removed from an agent group — it would behave in a similar fashion for place groups.

Here is the output from a typical request:

```
'EventCurrentTargetStateSnapshot' (17) attributes:
      TM_LENGTH [int] = 0
      USER_REQ_ID [int] = -1
      LONG_VALUE [int] = 0
      CURRENT_TARGET_STATE_INFO [CurrentTargetState] = CurrentTargetStateSnapshot (size=1) [
    [0] CurrentTargetStateInfo {
      AgentId = Analyst001
      AgentDbId = 101
      LoginId = null
      PlaceId = null
      PlaceDbId = 0
      Extensions = KVList:
'VOICE_MEDIA_STATUS' [int] = 0
'AGENT_VOICE_MEDIA_STATUS' [int] = 0
    }
  ]

      REQ_ID [int] = 5
      TM_SERVER [int] = 1245182089
```

Peeking at a Statistic

There may be times when you need to get immediate information on a statistic you have opened. For example, you may want to initialize a wallboard display. In that case, you can use `RequestPeekStatistic`. Note that Stat Server does not send a handshake event when you use this request, so you should use the send method rather than the request method when you use it. Note also that you need to use the `StatisticId` property to provide the `ReferenceId` of the `RequestOpenStatistic` or `RequestOpenStatisticEx` associated with the statistic you want information on:

> **Tip**
>
> If you use the request method on a `RequestPeekStatistic`, your request will time out and receive null, rather than retrieving the desired information from Stat Server.

```
[Java]

RequestPeekStatistic req = RequestPeekStatistic.create();
req.setStatisticId(2);

System.out.println("Sending:\n" + req);
statServerProtocol.send(req);
```

Suspending Notification

Because there are times when you do not need to collect information on a statistic for a while, the Platform SDK has requests that allow you to suspend and resume notification. These requests are like the peek request in that Stat Server does not send a handshake event when you use them, so you should use the send method rather than the request method when you use these requests. Note also that you need to use the `StatisticId` property of these requests to provide the `ReferenceId` of the `RequestOpenStatistic` or `RequestOpenStatisticEx` associated with the statistic you want information on. Here is how to suspend notification:

```
[Java]

RequestSuspendNotification req = RequestSuspendNotification.create();
req.setStatisticId(2);

System.out.println("Sending:\n" + req);
statServerProtocol.send(req);
```

Use code like this to resume notification:

```
[Java]

RequestResumeNotification req = RequestResumeNotification.create();
req.setStatisticId(2);

System.out.println("Sending:\n" + req);
statServerProtocol.send(req);
```

Closing the Statistic and the Connection

When you are finished communicating with Stat Server, you should close the statistics that you have opened and close the connection, in order to minimize resource utilization:

```
[Java]

RequestCloseStatistic req = RequestCloseStatistic.create();
req.setStatisticId(2);

System.out.println("Sending:\n" + req);
statServerProtocol.send(req);

...

statServerProtocol.beginClose();
```

.NET

Connecting to Stat Server

As mentioned in the article on the architecture, the Platform SDKs uses a message-based architecture to connect to Genesys servers. This section describes how to connect to Stat Server, based on the material in the article on Connecting to a Server.

After you have set up `using` statements, the first thing you need to do is create a `StatServerProtocol` object:

```
[C#]

StatServerProtocol statServerProtocol =
       new StatServerProtocol(new Endpoint(statServerUri));
statServerProtocol.ClientId = clientID;
statServerProtocol.ClientName = clientName;
```

You can also configure your ADDP and warm standby settings at this point, as described in the Connecting to a Server article.

Once you have finished configuring your protocol object, open the connection to Stat Server:

```
[C#]

statServerProtocol.Open();
```

Working with Statistics

The Stat Server application object in the Genesys Configuration Layer comes with many predefined statistics. You can also define your own statistics using the options tab of this application object. The

Platform SDK allows you to get information about any of these statistics by using RequestOpenStatistic. There may be times, however, when you want your application to be able to create new types of statistics dynamically. The Platform SDK also supports this, with the use of RequestOpenStatisticEx.

This section will show you how to use RequestOpenStatistic to get information on a predefined statistic. After that, we will give an example of how to use RequestOpenStatisticEx.

The first thing you need to do to use RequestOpenStatistic is to create the request:

[C#]

```
var requestOpenStatistic = RequestOpenStatistic.Create();
```

Now you need to describe the *statistics object*, that is, the object you are monitoring. This description consists of the object's Configuration Layer ID and object type, and the tenant ID and password:

[C#]

```
requestOpenStatistic.StatisticObject = StatisticObject.Create();
requestOpenStatistic.StatisticObject.ObjectId = "Analyst001";
requestOpenStatistic.StatisticObject.ObjectType = StatisticObjectType.Agent;
requestOpenStatistic.StatisticObject.TenantName = "Environment";
requestOpenStatistic.StatisticObject.TenantPassword = "";
```

Next, you will specify the StatisticMetric property for this statistic. A StatisticMetric contains information including the StatisticType (which must correspond to the name of the statistic definition that appears in the options tab), along with the required TimeRangeLeft and TimeRangeRight parameters.

In this case, we are asking for the total login time for an agent identified as "Analyst001":

[C#]

```
requestOpenStatistic.StatisticMetric = StatisticMetric.Create();
requestOpenStatistic.StatisticMetric.StatisticType = "TotalLoginTime";
requestOpenStatistic.StatisticMetric.TimeProfile = "Default";
// Note: if no time profile is provided, then the default is used automatically
```

Finally, specify the desired Notification settings. The Statistics Platform SDK supports four ways of gathering statistics:

- NoNotification allows you to retrieve statistics when you want them.
- Periodical means Stat Server reports on statistics based on the time period you request.
- Immediate means Stat Server reports on statistics whenever a statistical value changes. For time-related statistics, Immediate means that Stat Server will report the current value whenever a statistical value changes, but it will also report that value periodically, using the specified notification frequency.
- Reset means Stat Server reports the current value of a statistic right before setting the statistical value to zero (0).

In this case, we are interested in receiving statistics on a regular basis, so we have asked for a notification mode of Periodical, with updates every 5 seconds. For more information on notification modes, see the section on Notification Modes in Stat Server 8.5 User's Guide.

[C#]

```
requestOpenStatistic.Notification = Notification.Create();
requestOpenStatistic.Notification.Mode = NotificationMode.Periodical;

requestOpenStatistic.Notification.Frequency = 5; // seconds
```

Before sending this request, you have to assign it an integer that uniquely identifies it, so that Stat Server and your application can easily distinguish it from other sets of statistical information. Note that you will also need to enter this integer in the `StatisticId` field for any subsequent requests that refer to the statistics generated on the basis of the Open request.

> **Tip**
>
> `ReferenceId` is a unique integer that is specified for identification of requested statistics. If no value is set then this property is assigned automatically just before sending a message; however, if the property has already been assigned then it will not be modified. If you specify this property on your own, you should guarantee its uniqueness. StatServer's behavior was corrected (starting from releases 8.1.000.44 and 8.1.200.14 in corresponding families) so that if two requests are sent with the same `ReferenceId` then an `EventError` message is returned for the second request.

```
[C#]

requestOpenStatistic.ReferenceId = 3; // Must be unique and is included as StatisticId in
// Peek/Close for the stat
```

Now you can send the request:

```
[C#]

Console.WriteLine("Sending:\n{0}", requestOpenStatistic);
var response =
        statServerProtocol.Request(requestOpenStatistic);
Console.WriteLine("Received:\n{0}", response);

if (response == null || response.Id != EventStatisticOpened.MessageId)
{
        // Open failed, proper error handling goes here
        throw new Exception("RequestOpenStatistic failed.");
}

var @event = response as EventStatisticOpened;
```

After Stat Server sends the `EventStatisticOpened` in response to this request, it will start sending EventInfo messages every 5 seconds. You need to set up an event handler to receive these messages, as discussed in the the Event Handling article.

This is what one such message might look like:

```
[C#]

'EventInfo' ('2')
message attributes:
REQ_ID [int]    = 4
USER_REQ_ID [int] = -1
TM_SERVER [int] = 1244412448
```

```
TM_LENGTH [int] = 0
LONG_VALUE [int] = 0
VOID_VALUE [object] = AgentStatus {
        AgentId = Analyst001
        AgentStatus = 23
        Time = 1240840034
        PlaceStatus =   PlaceStatus = 23
        Time = 1240840034
        LoginId = LoggedOut
}
```

Creating Dynamic Statistics

As mentioned above, there may be times when you want to get statistical information that has not already been defined in the Configuration Layer. In cases like that, you can use RequestOpenStatisticEx. Before you do, however, you should make sure you understand several topics covered in the Reporting Technical Reference 8.0 Overview and the Stat Server 8.5 User's Guide, including the use of masks.

The first things you need to do in order to use RequestOpenStatisticEx are similar to what we did in the previous section. You will start by creating the request and specifying the statistic object and notification mode:

```
[C#]

var req = RequestOpenStatisticEx.Create();

req.StatisticObject = StatisticObject.Create();
req.StatisticObject.ObjectId = "Analyst001";
req.StatisticObject.ObjectType = StatisticObjectType.Agent;
req.StatisticObject.TenantName = "Resources";
req.StatisticObject.TenantPassword = "";

req.Notification = Notification.Create();
req.Notification.Mode = NotificationMode.Immediate;
req.Notification.Frequency = 15;
```

Now, instead of requesting a statistic type, you need to set up your own masks, as described in the section on "Metrics: Their Composition and Definition" in the Reporting Technical Reference 8.0 Overview. The following mask and statistic metric settings give the Current State for the agent mentioned above:

```
[C#]

var mainMask = ActionsMask.CreateDnActionMask();
mainMask.SetBit(DnActions.WaitForNextCall);
mainMask.SetBit(DnActions.CallDialing);
mainMask.SetBit(DnActions.CallRinging);
mainMask.SetBit(DnActions.NotReadyForNextCall);
mainMask.SetBit(DnActions.CallOnHold);
mainMask.SetBit(DnActions.CallUnknown);
mainMask.SetBit(DnActions.CallConsult);
mainMask.SetBit(DnActions.CallInternal);
mainMask.SetBit(DnActions.CallOutbound);
mainMask.SetBit(DnActions.CallInbound);
mainMask.SetBit(DnActions.LoggedOut);
```

```
var relMask = ActionsMask.CreateDnActionMask();

req.StatisticMetricEx = StatisticMetricEx.Create();
req.StatisticMetricEx.Category = StatisticCategory.CurrentState;
req.StatisticMetricEx.IntervalLength = 0;
req.StatisticMetricEx.MainMask = mainMask;
req.StatisticMetricEx.RelativeMask = relMask;
req.StatisticMetricEx.Subject = StatisticSubject.DNStatus;
```

Once you have set up the masks and the statistic metric, you can create a `ReferenceId` and send the request:

```
[C#]

req.ReferenceId = referenceIdFromRequestOpenStatistic;

Console.WriteLine("Sending:\n{0}", req);
var response =
        statServerProtocol.Request(req);
Console.WriteLine("Received:\n{0}", response);
```

Current Target State Events

You can use `RequestGetStatisticEx` and `RequestOpenStatisticEx` to set up the same type of current target state definitions that Universal Routing Server (URS) uses. (You can also set these up using Configuration Manager.) When this type of request has been sent, Stat Server sends some additional event types:

- `EventCurrentTargetStateSnapshot`
- `EventCurrentTargetStateTargetUpdated`
- `EventCurrentTargetStateTargetAdded`
- `EventCurrentTargetStateTargetRemoved`

The `Snapshot` event is returned in response to the open, while the `Updated` event is sent as state changes occur. In a situation where you open a `CurrentTargetState`-based statistic against an agent group, the `Added` and `Removed` messages occur when an agent is added to or removed from an agent group — it would behave in a similar fashion for place groups.

Here is the output from a typical request:

```
'EventCurrentTargetStateSnapshot' (17) attributes:
      TM_LENGTH [int] = 0
      USER_REQ_ID [int] = -1
      LONG_VALUE [int] = 0
      CURRENT_TARGET_STATE_INFO [CurrentTargetState] = CurrentTargetStateSnapshot (size=1) [
    [0] CurrentTargetStateInfo {
      AgentId = Analyst001
      AgentDbId = 101
      LoginId = null
      PlaceId = null
      PlaceDbId = 0
      Extensions = KVList:
'VOICE_MEDIA_STATUS' [int] = 0
'AGENT_VOICE_MEDIA_STATUS' [int] = 0
```

```
    }
  ]

      REQ_ID [int] = 5
      TM_SERVER [int] = 1245182089
```

Peeking at a Statistic

There may be times when you need to get immediate information on a statistic you have opened. For example, you may want to initialize a wallboard display. In that case, you can use `RequestPeekStatistic`. Note that Stat Server does not send a handshake event when you use this request, so you should use the `Send` method rather than the `Request` method when you use it. Note also that you need to use the `StatisticId` property to provide the `ReferenceId` of the `RequestOpenStatistic` or `RequestOpenStatisticEx` associated with the statistic you want information on:

> **Tip**
>
> If you use the `Request` method on a `RequestPeekStatistic`, your request will time out and receive null, rather than retrieving the desired information from Stat Server.

```
[C#]

var requestPeekStatistic = RequestPeekStatistic.Create();
requestPeekStatistic.StatisticId = 3;

Console.WriteLine("Sending:\n{0}", requestPeekStatistic);
statServerProtocol.Send(requestPeekStatistic);
```

Suspending Notification

Because there are times when you do not need to collect information on a statistic for a while, the Platform SDK has requests that allow you to suspend and resume notification. These requests are like the peek request in that Stat Server does not send a handshake event when you use them, so you should use the send method rather than the request method when you use these requests. Note also that you need to use the `StatisticId` property of these requests to provide the `ReferenceId` of the `RequestOpenStatistic` or `RequestOpenStatisticEx` associated with the statistic you want information on. Here is how to suspend notification:

```
[C#]

var requestSuspendNotification = RequestSuspendNotification.Create();
requestSuspendNotification.StatisticId = 3;

Console.WriteLine("Sending:\n{0}", requestSuspendNotification);
statServerProtocol.Send(requestSuspendNotification);
```

Use code like this to resume notification:

```
[C#]

var requestResumeNotification = RequestResumeNotification.Create();
requestResumeNotification.StatisticId = 3;

Console.WriteLine("Sending:\n{0}", requestResumeNotification);
statServerProtocol.Send(requestResumeNotification);
```

Closing the Statistic and the Connection

When you are finished communicating with Stat Server, you should close the statistics that you have opened and close the connection, in order to minimize resource utilization:

```
[C#]

var requestCloseStatistic = RequestCloseStatistic.Create();
requestCloseStatistic.StatisticId = 3;

Console.WriteLine("Sending:\n{0}", requestCloseStatistic);
statServerProtocol.Send(requestCloseStatistic);

...

statServerProtocol.BeginClose();
```