Demand Technology Software's Data Services SDK

Version 1.0

August 28, 2012

1. Overview

This document describes the Demand Technology Software (DTS) Data Services SDK. The SDK will allow 3rd Party applications to retrieve historical performance data from the Microsoft SQL Server-based Performance Sentry Performance Database (PDB) and utilize the data in .NET applications.

2. Architecture

The SDK consists of the **DTS.Common.DataAccess.Services** assembly which implements the **PerformanceDataService** class using C# in .NET 4.0. This class connects to the PDB to retrieve the requested performance data either by using LINQ or dynamic SQL queries. The class includes methods to get machine, performance object, and performance counter information. Connection management follows the .NET connection management paradigm for the **SQLConnection** class.

2.1. Dependencies

The data service assembly is built with .NET 4.0 and the following dependencies on related DTS common services.

Assembly Name	**Description**
DTS.Common.Collections	DTS common collection classes used for data requests.
DTS.Common.PerformanceData	Contains classes that define the Windows performance data, including PerformanceObject and PerformanceCounter. Also contains the utility function SQLIdentiferTransformer.
DTS.Common.Utils	DTS utility classes to support DTS.Common.Performance Data

These additional .dlls are included in the SDK.

3. Performance Data Service

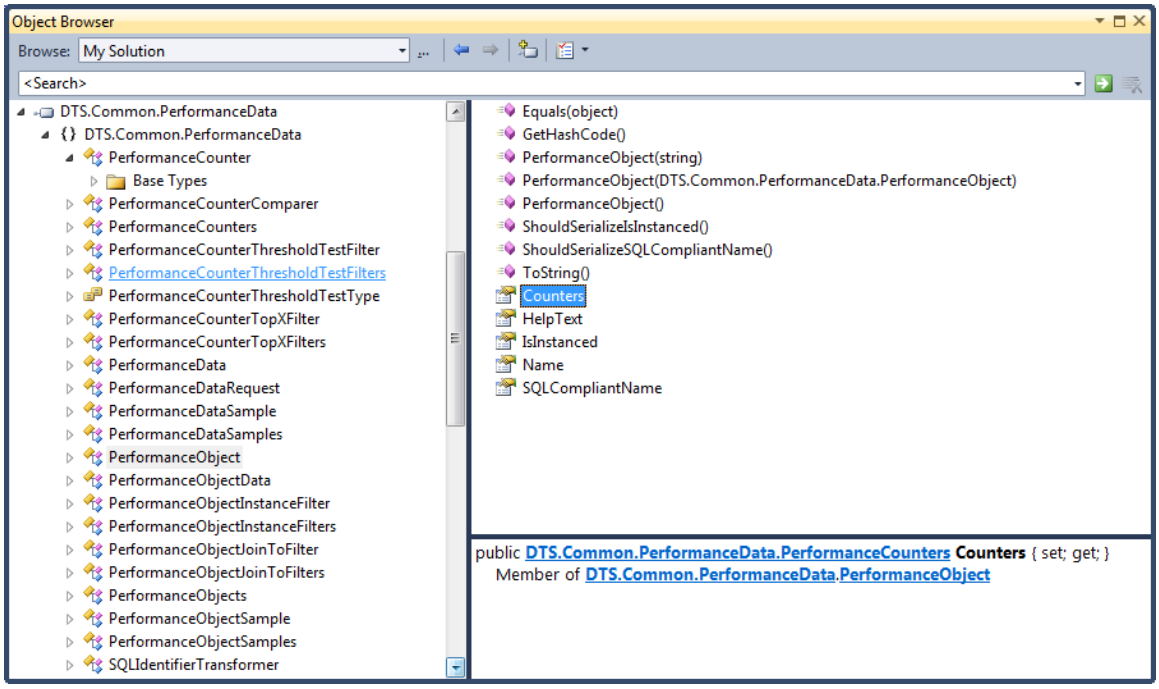
The bulk of the work to access performance data stored in the PDB is performed by instantiating the **PerformanceDataService** class. The public methods for the **PerformanceDataService** class are described below.

Method	Description
`public PerformanceDataService(string connectionString)`	The database connection string is specified in the constructor of the class and then connection management is performed by .NET in each subsequent method call.
`public List<DTS.Common.PerformanceData.PerformanceData> GetPerformanceData(PerformanceData.PerformanceDataRequest DataRequest)`	This function passes a **DataRequest** object which defines the data to be returned in a List<**PerformanceData**>
`public List<Machine> GetAllMachines()`	Returns all machines that have data in the PDB.
`public PerformanceCounters GetCounters(string machineName, PerformanceObject Object)`	This method returns counters that have data for a particular object by specifying the machine name either as a string or a class generated automatically from the PDB schema using dbml.
`public PerformanceCounters GetCounters(Machine machine, PerformanceObject Object)`	Same as above method but machine name is specified using a class generated from the PDB schema.
`public PerformanceObjects GetCollectionObjects(Machine machine)`	Returns the collected objects for a particular machine.

Performance data is retrieved by passing a **PerformanceDataRequest** object to the **GetPerformanceData** method. The **PerformanceDataRequest** object requires a start time, end time, and a list of counters to be returned over the requested time period. The objects will contain the desired performance counters.

PerformanceDataRequest
-ObjectsRequested : PerformanceObjects -InstanceFilters : PerformanceObjectInstanceFilters -PerformanceCounterThresholdTestFilters : PerformanceCounterThresholdTestFilters -PerformanceCounterTopXFilters : PerformanceCounterTopXFilters -JoinToFilters : JoinToFilters -StartTime : DateTime -Endtime : DateTime

The **PerformanceData** class contains **PerformanceObject** and **PerformanceCounter** classes from the **DTS.Common.PerformanceData** assembly to indicate the data to be retrieved. A **PerformanceObject** class contains a Counters property which is defined as an *ObservableCollection* of **PerformanceCounters**.



3.1. Valid SQL Identifiers

To deal with the Windows Performance object names and counter names that can contain invalid SQL characters, the PDB transforms the counter names into valid SQL identifiers. In the database, the names of counters that identify the database Table data columns are SQL-compliant.

To retrieve performance data and return a counter name that maps to the counter name displayed in the Windows Performance Monitor application, a reverse transformation is performed on the SQL column name by the **PerformanceDataService**. When submitting requests for performance counter data, you use the original object or counter name, which is transformed into a corresponding SQL-compliant column name.

For information purposes, the **DTS.Common.PerformanceData** classes also expose a public string Property called the **SQLCompliantName**, which is generated dynamically at run time from the object or counter name.

3.2. Unique Instance Names

The Windows Performance counters provide for an Instance name to identify counters associated with a specific processor, disk, etc., using a unique identifier. Following object-oriented terminology, this unique identifier is referred to as the Instance name.

The Instance names for the **Process** performance object are derived from the name of the process executable file name, i.e., sqlservr for the sqlservr.exe process. Windows allows multiple processes with the same executable name to run concurrently, so Instance name

is not sufficient to identify an instance of the Process performance object uniquely. The Process ID (or PID) is guaranteed to be unique at any one point in time, so the Process ID is used to distinguish among processes running the same executable.

To generate unique instance names for Process objects, the value of the ID Process counter is appended to the Instance name (delimited by a period: as in sqlsrvr.1113). If the user fails to request the ID Process counter for the **Process** object, it is added to the request automaticaly by the data service.

3.3. Performance Data Filters

There are several types of filters than can be specified in the data request to reduce the amount of data returned to the caller. Requests for data for Performance Objects with multiple instances, such as requests for Process level data, can return large volumes of raw performance data. Filtering reduces the amount of data that the PDB query returns.

Four types of filters are supported:

Filter	Description
InstanceFilter	Instance filters can be specified to restrict the data being returned to specific named instances of the objects requested.
JoinToFilter	This allows the request to return an instance of an object where the JOIN TO field matches the instances of another object, by time stamp. The JoinTo Filter is most frequently used to JOIN to the Process object to gather process-level processor and memory usage data. The JOIN operation is frequently performed on the Process ID field, serving as a foreign key. For example, the Performance Sentry collection agent adds an ID Process counter to each instance of the SQL Server performance objects that it gathers. Similarly, many of the .NET performance counters, such as .NET CLR Memory, contains the Process ID counter.
PerformanceCounterThresholdTest	Counter test filters can also be specified to return only instances of objects for which the specified counter meets the test.
PerformanceCounterTopXFilter	This return the top X samples of the counter for a particular object.

Multiple filters as well as multiple instances of each single filter can be specified, as required.

Use of the filtering capability is illustrated in the sample described in the following section.

4. Sample Code

The SDK provides sample code for a generic PDB Query desktop application that illustrates the use of the PDB Data Access services. The sample program is written in C# using Visual Studio 2010 and was built using the .NET Framework 4.0.

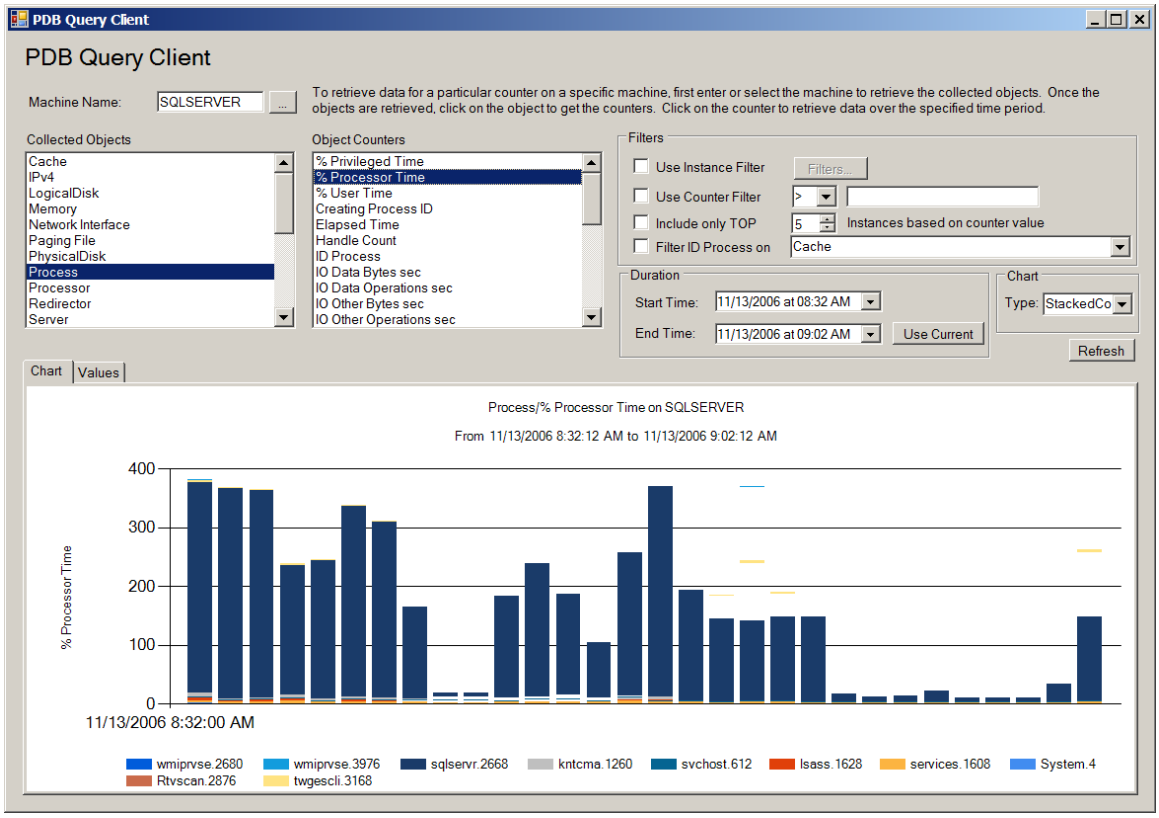
Project Name	Description	APIs Exercised	Language
PDBQueryClient	Sample client UI application for performance data retrieval.	*GetCollectionObjects* *GetPerformanceCounters* *GetPerformanceData*	C#

4.1. Building a UI Application

The SDK includes a sample application called **PDBQueryClient** to demonstrate how a custom data retrieval application might work. To retrieve historical data for a particular machine, the user can browse the network or type in a machine name. You specify the PDB ConnectionString setting to use in the PDBQueryClient.exe.config settings file:

```
    <applicationSettings>
        <DTS.Common.DataAccess.Services.Example.Properties.Settings>
            <setting name="ConnectionString" serializeAs="String">
                <value>Data Source=machine\database;Initial
Catalog=PDB;Integrated Security=SSPI;</value>
            </setting>
        </DTS.Common.DataAccess.Services.Example.Properties.Settings>
    </applicationSettings>
```

Once a machine is selected by the user, the collected objects on the machine are retrieved by calling the **GetCollectedObjects** method. Once the object is the selected, the program retrieves the available counters for the object on the selected machine by calling the **GetCounters** method. When a counter name is selected, a data request for that counter is created for the selected duration. The data request is then passed to the **GetPerformanceData** method on the **PerformanceDataServices** class. The results are shown on the chart and values tab. The sample also allows the user to test the four filter types, instance, counter threshold, top X, and JoinTo. A sample screen showing results after data is requested is shown below:



4.1.1. Making the Data Request

In order to request data, a **PerformanceDataRequest** object is created. The data request has an object list for which historical data is to be retrieved. These objects are instantiated and assigned a unique Instance name. Counters are assigned to the object for data that the user wants to retrieve. It is important to only request data which will be used in order to minimize bandwidth used for the request.

Note: Any object, counter, or instance names specified in the data request object are case-sensitive.

```
PerformanceDataRequest request = new PerformanceDataRequest();

// Set up the performance object
PerformanceObject po = new PerformanceObject(Object.Name);
PerformanceCounter counter = new PerformanceCounter(CounterName);
po.Counters.Add(counter);
```

If a list counters names of a particular object are needed, then the client can call the function **GetCounters** for a particular object and machine. This will return a list of the counters being collected for the object. This list can then be used to build the list of counters for the request.

```
// Create the performance object request list
request.ObjectsRequested = new PerformanceObjects();
request.ObjectsRequested.Add(po);
request.Machines = new string[] { tbMachineName.Text };
request.StartTime = this.StartTime.Value;
request.EndTime = this.EndTime.Value;
```

The sample code also demonstrates how to set up the optional filters for the request. Object instance filters can reduce the amount of data returned by restricting the query to specific instances of an object. In the example below, the Process object data is only returned for the instances specified.

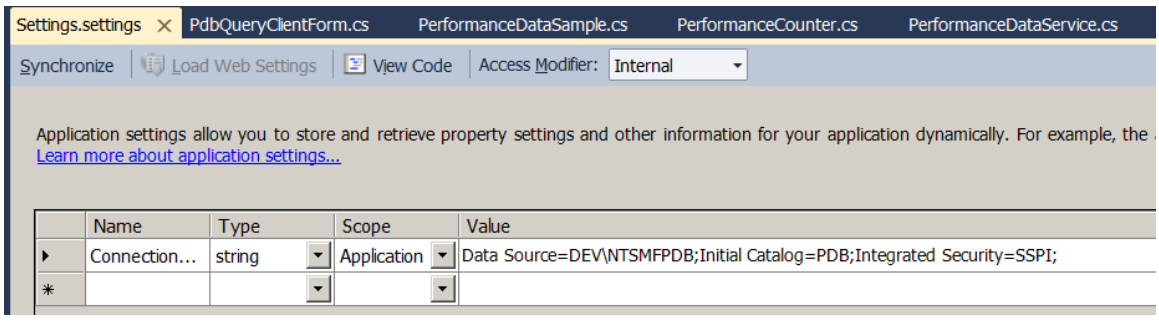
```
PerformanceObjectInstanceFilter instanceFilter = new
PerformanceObjectInstanceFilter();

instanceFilter.ObjectName = this.ObjectName;
instanceFilter.InstanceNames = new string[lbInstances.Items.Count];
for (int i = 0; i < lbInstances.Items.Count; i++)
instanceFilter.InstanceNames[i] = lbInstances.Items[i].ToString();
```

The user may not want to retrieve all of the available data for the object if the time interval is large. To further restrict the amount of data being returned, a counter filter exists to filter on a specific value of a counter in an object. The threshold can be a test whether the counter value is less than, less than or equal, greater than, greater than or equal, or just equal to a value.

An alternative to the threshold test counter filter is the TopX counter filter. This filter allows only the top X values to be returned for the counter over the specified time period.

Lastly, the JoinTo filter allows the developer to have instances of a particular object returned if one of the object's counters has the same value as one of another object's counters for a specific timestamp. This functionality is used when requesting process object data for a SQL Server object where the unique ID Process field is also available.

Once the request object is created, the data service can be opened using the specified SQL Connection string. For the sample application, the SQL Connection string is stored in the program's settings.



The connection string is retrieved from the application settings as follows:

```
string connectionString = Properties.Settings.Default.ConnectionString;
```

If a user name or password is needed, the connection string settings are **User ID** and **Password**. In the sample application, these are not used. For the sample program, the settings are stored in the **PDBQueryClient.exe.config** xml file.

Note: In a production environment, encryption should be when storing the Connection string information to protect access to the PDB.

Once the connection string is retrieved, then the data service can be instantiated.

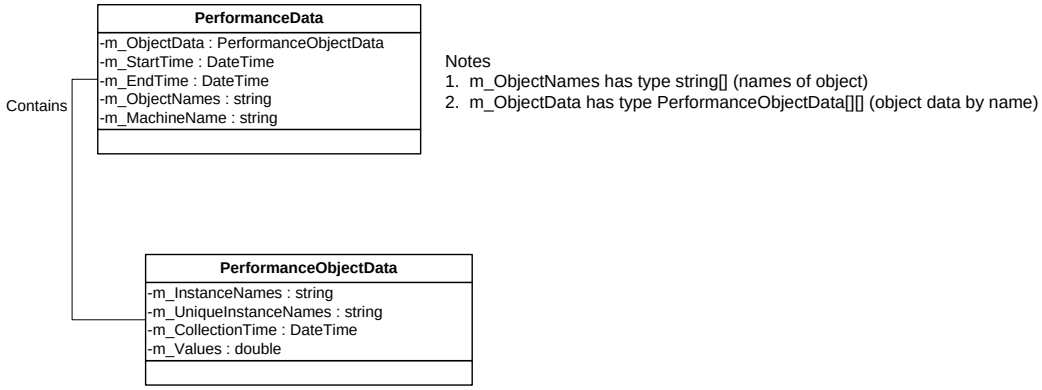
```
  service = new
DTS.Common.DataAccess.Services.PerformanceDataService(connectionString);
```

4.1.2. Examining the data

The data is returned from the **GetPerformanceData** call in a generic .NET collection, namely a List of **PerformanceData** objects.

```
      List<DTS.Common.PerformanceData.PerformanceData> result =
service.GetPerformanceData(request);
```

One **PerformanceData** object is returned for each one of the machines in the data request.



The **PerformanceData** class contains an **ObjectNames** field which is an array of strings with the names of the objects to fulfill for the request. If the first object request is 'Process', then ObjectNames[0] will be 'Process' The **ObjectData** field is the array (potentially, a sparse array) of **PerformanceObjectData** instantiations for each object. If

the first object requested is 'Process', then the PerformanceObjectData for the process will be in the array in the location of ObjectData[0].

The **PerformanceObjectData** class is utilized to hold the counter values of each instance returned for a particular requested object at a particular collection time. The names of the instances returned are in the **InstanceNames**[] string array. The name of the first instance of an object at a particular collection time indexed by the InstanceNames[0] value. Uninstanced objects will have a blank instance name.

The actual counter values are in a sparse array called **Values**. The counter values are indexed in the same order as they are requested. For the first instance of a particular object, the Values would be in the Values[0][0] array.

Note: Counter values must be checked for null values since uncollected counter values are represented by null instead of 0.0 or some distinct numerical value. See the highlighted text in the example below.

The following code constructs a message containing the object instances and values:

```
// Data is returned in the ObjectData array for each object name
for (int i = 0; i < data.ObjectNames.Length; i++)
{
  // Instances of the object are stored in the ObjectData array sorted by Time
  for (int instance=0; instance < data.ObjectData[i].Length; instance++)
  {
     // Get the sample for the instance of the ith object
     PerformanceObjectData sample = data.ObjectData[i][instance];

      string message = data.ObjectNames[i] + "," +
      sample.CollectionTime.ToString();

     // Each object has a number of instances
    for (int j=0; j < sample.InstanceNames.Length; j++)
    {
          message =  message + "\r\n" + sample.InstanceNames[j];
         // Each instance has a number of values ordered by the request order
         for (int k = 0; k < sample.Values[j].Length; k++)
         {
              message = message + ",";

              if (sample.Values[j][k] != null)
                   message += sample.Values[j][k];
         }
    }
}
```

In a graphical analysis program, it is easier to deal with a list of **PerformanceDataSample** objects instead of the packed data structures that are used to communicate directly with the PDB. The SDK includes a **ToSamples** method in the **PerformanceData** object to do this transformation. The method will transform the values for an object and counter in an array of **PerformanceSample** objects. Each sample contains the value of the particular object and counter for one collection interval.

```
public void ToSamples(PerformanceObject po,
   string Counter,
   PerformnaceDataSamples dataSamples)
```

PerformanceDataSample
-ObjectName : string -CounterName : string -UniqueInstanceName : string -InstanceName : string -CollectionTime : DateTime -DValue : double -Value : string

Once the data is packaged into the **PerformanceDataSamples** object, the data samples can be accessed by an application which binds the data from one of the .NET collection classes to a data-bound reporting control.

4.1.3. Graphing the Result

The PDBQueryClient application uses the .NET Chart control to display the performance data. The first step is to request the data and transform the results into the **PerformanceDataSamples** object using the **ToSamples** method.

```
       // List of performance data on each machine requested
      List<DTS.Common.PerformanceData.PerformanceData> result =
service.GetPerformanceData(request);
      if ((result != null) && (result.Count > 0))
      {
         // Transform the data into performance data samples for the requested
object and counter
         result[0].ToSamples(po,
                    Counter.Name,
                     dataSamples);

                    ….
```

Once the data is transformed to the **PerformanceDataSamples** object, it can be added to the graphing control by iterating over the collection and adding the information from each **PerformanceDataSample** individually to the data chart series for that instance of the object.

```
// There is one data sample for each instance at a particular time
for (int i = 0; i < dataSamples.Count; i++)
{
         PerformanceDataSample sample = dataSamples[i] as PerformanceDataSample;

         // Add one series to the chart for each instance of the object requested
         Series s = null;

         // use 'uninstanced' for the instance name
         string InstanceName = "";
```

```
            if (sample.UniqueInstanceName.Length == 0)
               InstanceName = "Uninstanced";
            else
               InstanceName = sample.UniqueInstanceName;

            s = this.DataChart.Series.FindByName(InstanceName);
            if (s == null)
            {
                 s = new Series(InstanceName);
                 s.XValueType = ChartValueType.DateTime;
                 this.DataChart.Series.Add(s);
            }

            ...

            // Add the point CollectionTime,Value
            if ((s != null) && (s.Points != null))
            {
                int iAdded = s.Points.AddXY(sample.CollectionTime, sample.DValue);
                if (iAdded >= 0)
                {
                     DataPoint dp = s.Points[iAdded];
                     dp["TTlabel"] = sample.CollectionTime.ToShortTimeString();
                }
            }

        ...
}
```

This document and sample code explains how to retrieve data from the Performance Database (PDB) using the DTS Data Services SDK. It demonstrates how to prepare a data request, examine the data, and graphing the end result.

Feedback or questions can be sent to support@demandtech.com