Appeon Performance Tuning Guide

Appeon® for PowerBuilder® 2013 R2

FOR WINDOWS & UNIX & LINUX

Contents

1 About This Book

1.1 Audience

This book is intended to help PowerBuilder developers plan what steps they will take, and how much time they will invest in improving the performance of a PowerBuilder application deployed to the Web/Mobile with Appeon for PowerBuiler. It is also intended to guide PowerBuilder architects and developers on how to build new PowerBuilder applications that perform well when deployed to the Web, a WAN or the Mobile.

1.2 How to use this book

There are eight chapters in this book.

Chapter 1: About This Book

A general description of this book.

Chapter 2: Appeon Performance

Describes current runtime performance levels of Appeon for PowerBuilder and primary reasons for performance issues (if any).

Chapter 3: Identifying Performance Bottlenecks

Describes approach for identifying areas in the Web application that may suffer from runtime performance issues. Also, describes the Appeon Developer performance reporting tool - the Appeon Performance Analyzer tool.

Chapter 4: Performance-Related Settings

Documents key Appeon, Web browser, application server, and database settings that should be configured from default values to ensure optimal performance of your Web/Mobile system.

Chapter 5: Tuning: Excessive Server Calls

Introduces the performance tuning concept "Excessive Server Calls" with several techniques to optimize your PowerBuilder code to achieve good performance.

Chapter 6: Tuning: Heavy Client

Introduces the performance tuning concept "Heavy Client" with several techniques to optimize your PowerBuilder code to achieve good performance.

Chapter 7: Tuning: Large Data Transmissions

Introduces the performance tuning concept "Large Data Transmissions" with several techniques to optimize your PowerBuilder code to achieve good performance.

Chapter 8: Conclusion

Final thoughts and recommendations on performance tuning of your PowerBuilder applications for the Web/Mobile.

1.3 Related documents

Appeon provides the following user documents to assist you in understanding Appeon for PowerBuilder and its capabilities:

- Introduction to Appeon:

  Gives general introduction to Appeon for PowerBuilder and its editions.

- Getting Started (for Appeon Mobile):

  Guides you though installing PowerBuilder and Appeon for PowerBuilder, and developing and deploying a mobile application.

- New Features Guide:

  Introduces new features and changes in Appeon for PowerBuilder.

- Appeon Mobile Tutorials:

  Gives instructions on deploying, running, and debugging the mobile application, distributing native mobile apps, and configuring the Appeon Server cluster.

- Appeon Mobile (Offline) Tutorials:

  Gives instructions on setting up the Appeon Mobile (Offline) environment, and configuring, deploying, running, updating, and debugging the offline application.

- Appeon Installation Guide:

  Provides instructions on how to install Appeon for PowerBuilder successfully.

- Installation Guide on Cloud Platform:

  Provides instructions on how to install Appeon for PowerBuilder on the cloud-based platform such as Windows Azure and AWS EC2 and S3.

- Mobile UI Design & Development Guide:

  Introduces general guidelines on designing and developing the mobile app and UI.

- Migration Guidelines for Appeon Web:

  A process-oriented guide that illustrates the complete diagram of the Appeon Web migration procedure and various topics related to steps in the procedure, and includes a tutorial that walks you through the entire process of deploying a small PowerBuilder application to the Web.

- Supported PB Features:

  Provides a detailed list of supported PowerBuilder features which can be converted to the Web/Mobile with Appeon as well as which features are unsupported.

- Appeon Developer User Guide:

  Provides instructions on how to use the Appeon Developer toolbar in Appeon for PowerBuilder.

- Workarounds & API Guide:

  Provides resolutions for unsupported features and various APIs to facilitate you to implement the features (including Web and mobile) that are not easy or impossible to implement in the PowerBuilder IDE.

- Appeon Workspace User Guide:

  Gives a general introduction on Appeon Workspace and provides detailed instructions on how to use it.

- Appeon Server Configuration Guide:

  Provides instructions on how to configure Appeon Server Monitor, establish connections between Appeon Servers and database servers, and configure AEM for maintaining Appeon Server and the deployed applications.

- Web Server Configuration Guide:

  Describes configuration instructions for different types of Web servers to work with the Appeon Server.

- Troubleshooting Guide:

  Provides information on troubleshooting issues; covering topics, such as product installation, application deployment, AEM, and Appeon application runtime issues.

- Appeon Performance Tuning Guide:

  Provides instructions on how to modify a PowerBuilder application to achieve better performance from its corresponding Web/mobile application.

- Testing Appeon Web Applications with QTP:

  Provides instructions on how to test Appeon Web applications with QTP.

1.4 If you need help

If you have any questions about this product or need assistance during the installation process, access the Technical Support Web site at http://www.appeon.com/support.

2 Appeon Performance

Before we officially introduce the methods to analyze and optimize the performance, we spare a few minutes to look at the possible impacts that cause performance problems to Appeon-converted applications. This helps to learn optimization solutions better, or even helps to work out individual optimization solutions for each application.

Traditional PowerBuilder application architecture

The traditional PowerBuilder application is based on the client/server architecture, in which applications run on the client side and interact with the database with native drivers. The connection between the client and the database (marked as "A" in the following figure) is usually the local area network or high-speed enterprise network, therefore the connection can hardly impact the performance. In this architecture, the main performance impacts are the application and the database.

Figure 2.1: Traditional PB application architecture
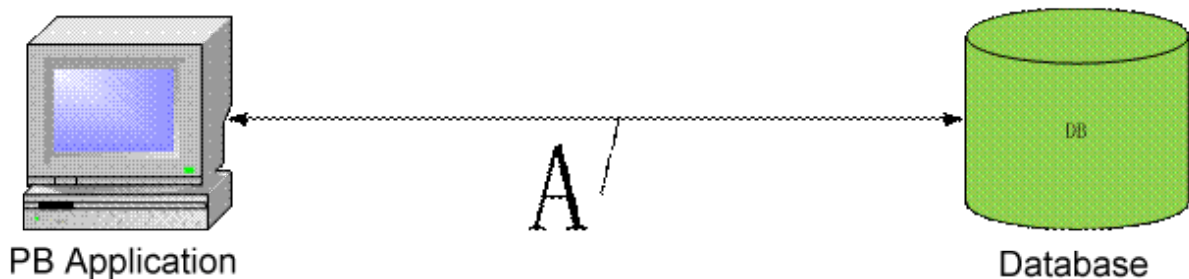
Thus, the performance of C/S PB applications = PB client performance + A + database processing performance.

Since the connection impact can be negligible, the performance of C/S PB applications = PB client performance + database processing performance.

Appeon (Web/Mobile) application architecture

Compared with the traditional C/S architecture, the Appeon application architecture has a server (that holds the core business logic and the data service) between the client and the database. The client does not directly interact with the database anymore, but sends all the requests and data to the server, and interacts with the database through the server.

Figure 2.2: Appeon application architecture
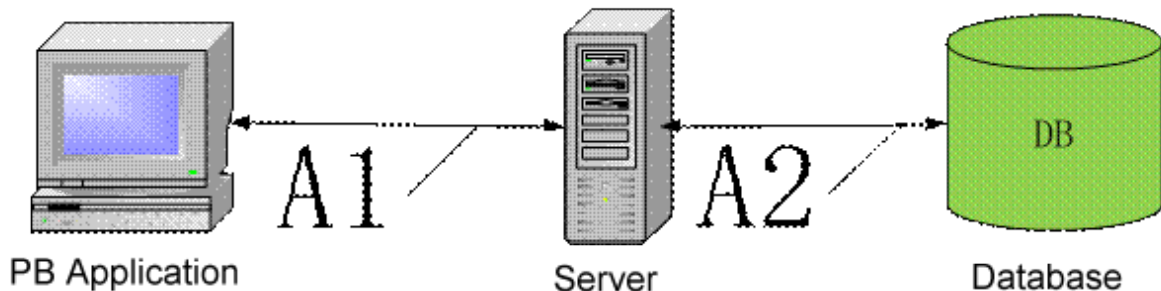
In the architecture, the Appeon application performance = Web/Mobile client performance + A1 + server performance + A2 + database processing performance.

A1 is the network connection between the client and the server, and is usually the wide area networks/internet. Therefore, this is a very large performance expense, especially when the client frequently interacts with the server.

A2 is the network connection between the server and the database, and is usually the local area networks or high-speed enterprise networks. The same as that in the C/S architecture, it can hardly impact the performance.

Therefore, the Appeon application performance = Web/Mobile client performance + A1 + server performance + database processing performance.

PowerBuilder application impacts vs. Appeon application impacts

According to the PowerBuilder application architecture and the Appeon Web/Mobile application architecture described above, we know that:

- PB application performance = PB client performance + database processing performance

- Appeon application performance = Web/Mobile client performance + A1 + server performance + database processing performance

Appeon applications are converted from PB applications and are consisted of two parts after migration: the Web/Mobile client and the server. Therefore, the Appeon application performance is dependant on the PB application performance.

Two hypotheses as follows:

1. The PB application does not have performance problems; but the converted Appeon application has.

   In this case, the PB client and the database performance expense are negligible. Since the performance of Appeon Web/Mobile client and the server are dependent on the PB client performance, the performance problem may be caused by A1, the connection network.

   And the possible reasons are:

   - The networks connection is slow or unstable;

   - The data package is too large or the SQL syntaxes are not efficient that result in long communication time in a single communication;

   - The same functionality frequently communicates with the server that results in repeated connection performance (A1) expense, etc.

   In the case, developers should: 1) first consider to reduce the communication times between the client and the server so to reduce the connection performance expense; 2) secondly, consider to optimize the efficiency of each communication, for example, by retrieving only the necessary data and using the optimal relational calculus in the SQL syntaxes, etc.

   Usually, if the PB application does not have performance problems, the Appeon application converted from it will not have problems either. But under certain circumstances, there still can be performance problems in the converted Appeon application. This is because the PB application will finally run using the machine code or the quasi machine code, which is very efficient at code levels; but the converted Appeon

application executes the JavaScript, an interpreted code, and will interpret at runtime. Therefore, the efficiency is lower compared with the machine code, especially when there is

- Functionalities that use multiple character string manipulation;

- Lots of loops;

- Frequently-distributing memories or large memories, for example, large objects, lots of tabpages in the tab control, etc.

2. Both the PB application and the converted Appeon application have performance problems:

   If the PB application has performance problems, the converted Appeon applications will definitely have performance problems as well.

   In this case, developers should:

   1) first consider to optimize the performance of the PB application and the database by using all kinds of available system tools. For example, developers can use the transaction track analyser provided by the database provider to analyze and optimize the database performance. Usually, popular database providers provide performance analysis and optimization tools with their databases, developers can use these provided tools to optimize the databases.

   2) secondly, after you make sure that the PB application does not have performance problems, use the hypothesis 1 to analyze and the converted Appeon application.

In details, the impacts to the converted Appeon application are as follows:

- Impact of the Internet and slow networks;

- Impact of "heavy" client-side logic;

- Impact of large data transmission.

Refer to the Impacts to Appeon performance for more information.

2.1 Impacts to Appeon performance

2.1.1 Impact of the Internet and slow networks

Although Appeon pushes the envelope to deliver unparalleled performance from standard Web technologies (e.g. XML, JavaScript, HTML, Java or C#), which are typically significantly slower than PowerBuilder, slow and latent network connections rob performance from even the best applications!

Network chatter and network-intensive code really highlight the weakness of a poor network connection. Any code that results in a HTTP request (i.e. server call) when executed multiple times sequentially has potential to create network chatter. There are mainly two categories of code that result in server calls - data access related and remote method invocations. Here are several common examples so you can familiarize yourself:

- Embedded SQL (Select, Insert, Delete, Update, Cursor) including Dynamic SQL;

- Invoking stored procedures or database functions;

- DataWindow/DataStore Functions (Retrieve, Update, ReselectRow, GetFullState, SetFullState, GetChanges, SetChanges);

- DataWindow/DataStore Events (SQLPreview);

- Invoking a method of a server-side object, such as a PowerBuilder NVO, Java EJB, or .NET Component; or

- Invoking a Web Service.

Each of the above statements will generate one call to the server utilizing HTTP, with exception of SQLPreview event that will generate one call for each line of code handled by the event. If any of the above statements are contained in a loop or recursive function, well depending on the number of loops, even though its just one statement it would be executed multiple times generating multiple server calls. Needless to say, loops and recursive functions are some of the most dangerous from a performance perspective.

The reason it is important to minimize server calls is because it can take 100 or even 1,000 times longer to transmit one packet of data over the Internet compared to a LAN. Imagine an event handler is triggered, for example handling an "onClick" event, whose execution will result in 80 synchronous server calls over a LAN with latency of 2 milliseconds (ms). In such scenario the slow-down attributed to network latency would be 0.16 seconds (80 x 2 ms). Now imagine this same event handler running over a WAN with latency of 300 ms. The slow-down attributed to the network latency would be a whopping 24 seconds (80 x 300 ms)! And depending on the amount of data transmitted there could be additional slow-down due the bandwidth bottlenecks.

It is imperative for the developer to be conscious that PowerBuilder applications deployed to the Web may not be running in a LAN environment, and as such there will be some degree of performance degradation. How much depends on how the code is written, but in most cases the performance degradation still falls within acceptable limits without much performance optimization.

Should you find that certain operations in your application are unacceptably slow, the good news is there are numerous things that you can do as PowerBuilder developers to ensure your PowerBuilder applications perform well in a WAN environment (e.g. Internet) or on slower networks. At a high-level, your code needs to be written such that the server calls and other performance intensive code is minimized or relocated to the middle-tier or back-end. This will be covered in more detail in the following chapters. Some changes are actually quite simple while others may require increased effort. Nonetheless, in all cases optimizing the performance of your applications in PowerBuilder is just a fraction of the effort to work with typical low-productivity Web tools such as VisualStudio.NET and Eclipse.

2.1.2 Impact of #heavy# client-side logic

Most PowerBuilder applications are developed utilizing a 2-tier architecture. In other words, all the PowerScript and embedded SQLs are coded in the Visual objects, for example Window, CommandButton, etc. In contrast, a 3-tier architecture would encapsulate all

non-visual logic in PowerBuilder NVOs (Non-Visual Objects). The reality is even if your application utilizes NVOs, chances are it is not a pure 3-tier application if PowerBuilder NVOs are not exclusively utilized to encapsulate all non-visual logic. But don't rush to partition your application just yet!

Most applications developed as a 2-tier architecture perform great in Appeon. In fact, there are many situations that a 2-tier application when deployed by Appeon will actually perform faster than a 3-tier application. The reason is if a PowerBuilder NVO is deployed to the middle-tier or application server, time must be spent to call the server and get the results back to the client. Of course, your non-visual logic running on an application server will run faster than at the Web browser. The key question is how much performance do you gain by running a particular block of code on the application server vs. how much performance do you lose due to the server calls.

As a rule of thumb, it is recommended to partition your non-visual logic to the middle-tier only when the particular block of code runs unacceptably slow at the Web browser. In such cases, it is likely that the application performance will benefit, and as such, it is worthwhile to invest the time to partition such logic. However, if the non-visual logic is only slightly sluggish, it may be possible to optimize the code without having to partition it to the application server.

2.1.3 Impact of large data transmission

When you first open a Window there are two types of files downloaded. The first type is the HTML and JavaScript files ("Web files") that contain the UI and UI logic of the application Window. The second type is data files that contain the result set, for example for a DataWindow retrieve. The time to download these files is affected by two factors: 1) the network connection and 2) the size of the files to be downloaded.

The Web files do not impact performance because of their small size and the enhanced ability of the browser to "cache" them. The Web files for a given PowerBuilder Window are typically between 25-75 KB. Because these Web files are static in nature, once a given application Window has been opened, the Web files will be cached on the Client computer. As such, once these Web files are cached, their impact on performance is essentially non-existent.

Under most circumstances, these Web files are not re-downloaded when the Window is reopened. The only exceptions are if 1) the temporary Internet files folder has been emptied or 2) the application has been updated and redeployed to the server. If the latter has happened, Web files for only those Windows that have been modified will be automatically downloaded from the Web server.

Only the data files containing the result sets may or may not be cached (depending on whether you have enabled DataWindow caching). A result set of 50 records would typically result in a 12 KB data file. Every 5 records would typically add another 1.2 KB to the data file size. So, for example, a 500-record result set would typically correspond to a 120 KB data file. If DataWindow caching is not enabled, the data files corresponding with such DataWindows will be downloaded from the server each time a DataWindow retrieve is invoked.

The good news is that Appeon has built-in 10X data compression for DataWindow result set to essentially eliminate the time spent downloading these data files. The same 500-record

result set that would normally correspond to a 120 KB data file would only result in the download of a 12 KB data file from the server. This compression feature makes even the largest of result sets quick to transfer.

In conclusion, due to Web file caching feature of the Web browser, Appeon's built-in DataWindow caching technology and 10X data compression technology, generally speaking neither the Web files nor data files should have any noticeable impact on the performance of your Web application.

2.2 Expected performance level

When comparing a PowerBuilder application to the performance of a Web application deployed by Appeon to a LAN environment, generally speaking the performance of the two will be quite similar. In some cases (for certain operations) Appeon may actually be even faster than PowerBuilder.

The reason is that Appeon has been tuned for nearly a decade to offer the best performance possible for real-life PowerBuilder applications:

• Large PowerBuilder applications up to 600MB (of PBLs) including several thousand DataWindows and thousands of Windows.

• Complex screens containing as much as 80 DataWindows in a single Window

• Complex tabs (including nested tabs and dynamically created tabs)

• Dynamically created objects (DataWindows, UserObjects, etc.)

• PFC and other high-overhead frameworks similar to PFC.

2.3 Automatic performance boosting

Appeon has a number of features built into its infrastructure/framework to instantly or automatically boost the performance of PowerBuilder applications when deployed to the Web. Many of these features are always on and transparently working in the background to boost performance. Other features are user-selectable and must be configured. The following table is a list of these features and the configuration of these features is covered in Chapter 3, Performance-Related Settings.

Table 2.1: Appeon Performance Boosting Features

Performance Feature	Description	Location
Just-in-Time Downloading	As the application is run and various windows are opened, only the Web files required for that particular window are downloaded at that point in time. Once the Web files are downloaded, they are cached in the Internet Explorer temporary files folder and are not downloaded again.	Appeon Infrastructure
10X Web File Compression	All JavaScript files are compressed by as much as 10X, then the compressed version of the file is downloaded over HTTP to the Web browser.	Appeon Developer

Performance Feature	Description	Location
10X Data File Compression	For each DataWindow or DataStore retrieval, the result set is first retrieved by the application server, automatically compressed by 10 times in most cases, and then downloaded over HTTP to the Web browser. Utilizing AJAX technology, only the DataWindow or DataStore is refreshed and the rest of the screen remains intact.	Appeon Infrastructure
DataWindow Data Caching	For each DataWindow or DataStore, the developer has the option of enabling caching of the result set. Appeon enables caching at the application server, Web server, and Web browser so every tier of the Web architecture is benefiting from the best performance and scalability possible.	Appeon Enterprise Manager
Merge files	Merges multiple JavaScript files into a single file to reduce the number of HTTP requests and corresponding overhead.	Appeon Developer
Multi-thread Downloading	Downloads are multi-threaded to boost the application runtime performance.	Appeon Enterprise Manager
Custom Libraries Downloading	Any custom libraries can be automatically downloaded and installed with your Web application, or if the libraries are very large in size, you can disable this feature and distribute the libraries some other fashion.	Appeon Enterprise Manager
Database Connection Pooling	By deploying to a true n-tier Web environment with Appeon, you can take advantage of Database Connection Pooling, a feature of most application servers. Connection Pooling does exactly that, it establishes a pool of connections to your database, which is shared among your clients. So rather than each client having its own dedicated connection to the database, a fewer number of connections can be rotated among all the users. For large deployments with thousands of clients this can boost database scalability noticeably.	Appeon Infrastructure

3 Performance-Related Settings

3.1 Overview

Performance settings need to be configured in Appeon Developer, Appeon Enterprise Manager, Web browser and your application server to ensure good performance in a production environment. If you identify any performance bottlenecks, it is strongly recommended that you first ensure all performance-related settings are correctly configured. Only if the performance issues still persist after the performance settings are configured correctly, then it is recommended to consider optimizing your PowerBuilder code.

3.2 Appeon Developer performance settings

The following table lists the Appeon performance settings that the developer can configure (in the application profile) of Appeon Developer to boost performance. To make a performance setting effective for an application, enable the option in the application profile and then perform a "Full Deployment" of the application.

Table 3.1: Performance settings in Appeon Developer

Performance Feature	Description	User-Selectable
10X Web File Compression	Compresses files when they are transferred over the network.	User must enable this feature for it to become effective.
Merge files	Merges the small files during the application deployment. The small files will be downloaded to the client in one file package at one call, instead of being downloaded one by one at separate calls.	User must enable this feature for it to become effective.
Performance Analyzer compliance	Enables the application to be deployed in a method compliant with the Performance Analyzer tool. The application deployed with this option can run with the Performance Analyzer tool at runtime to diagnose the performance issues. For more about the Performance Analyzer tool, see Appeon Performance Analyzer. Note: as the data is not encrypted during transmission at runtime, therefore, this option is recommended to use only at the test or development stage.	User must enable this feature for it to become effective.
Download ActiveX files in a single thread	Downloads the two ActiveX files in the same thread at runtime. Using the same thread to download the two ActiveX files can speed up the download under particular network conditions.	User must enable this feature for it to become effective.

3.3 Appeon Enterprise Manager performance settings

3.3.1 Timeout settings

By setting proper values for Timeout Detection Interval and Timeout Settings in AEM, Appeon Server can release the timeout and invalid database connections in time, thus can avoid database deadlock or malfunctions, so that the concurrent processing ability and the running stability of Appeon Server for .NET can be greatly enhanced.

3.3.1.1 Session Timeout Detection Interval

The Session Timeout Detection Interval setting is to specify the interval (in seconds) to check whether sessions have timed out.

It is recommended that the value is smaller than the Session Timeout value. The default value is 30 seconds. The value recommended by the system is based on your Session Timeout values, if the Session Timeout values are different from applications, the system will multiply the smallest value with 0.15 to get the recommended value.

3.3.1.2 Transaction Timeout Detection Interval

The Transaction Timeout Detection Interval setting is to specify the interval (in seconds) to check whether transactions have timed out.

It is recommended that the value is smaller than the Transaction Timeout value. The default value is 30 seconds. The value recommended by the system is based on your Transaction Timeout values, if the Transaction Timeout values are different from applications, the system will multiply the smallest value with 0.15 to get the recommended value.

3.3.1.3 Session timeout

Session timeout ends the user session and rolls back all database updates since the last commit for a user session. The default value is 3600 seconds. Session Timeout should be larger than the Transaction Timeout setting. Generally speaking, Session Timeout should not be smaller than 3600 seconds.

3.3.1.4 Transaction timeout

Transaction timeout rolls back all database updates since the last commit in a transaction. The default value is 120 seconds. Transaction Timeout should be less than Session Timeout.

If transaction timeout in the application database is set to 1800 seconds, then Transaction Timeout in AEM should be set to 1810 or larger.

If transaction timeout in the application database is not set, then Transaction Timeout in AEM should be set to a number greater than the maximum time needed to execute regular database operations for the Web application, suppose the most time-consuming table query operation takes 3600 seconds to complete, then Transaction Timeout should be set to 3610 or larger.

3.3.1.5 Rollback Completion time

Specifies the maximum time (in seconds) to complete a rollback of a transaction. If the time to roll back transaction exceeds the specified value, the transaction rollback will be terminated. The default value is 3 seconds.

It is recommended that it is set to a number between 3 seconds and Transaction Timeout *0.1.

3.3.1.6 Maximum Rollback Retries

The maximum times to retry a rollback of a transaction. Appeon Server executes a rollback if a transaction times out. If the rollback fails, Appeon Server will keep retrying until this maximum value is reached. Setting this value to 0 will disable this feature. The default value is 3.

It is recommended that it is set to a number smaller than 10.

3.3.2 DataWindow data caching

Appeon Enterprise Manager (AEM) provides a DataWindow data cache mechanism for caching the frequently used DataWindow data. It is recommended that you enable the cache for the DataWindow objects whose data is relatively static. Any DataWindow objects whose data is fairly dynamic should remain unchecked, otherwise you will experience overhead from the caching mechanism without the true benefits of the caching.

NOTE: DataWindow Data Cache will not be effective until you fulfill all the configuration requirements that are detailed in Section 5.4.8.3, "DataWindow Data Cache" in *Appeon Server Configuration Guide for .NET* or in Appeon Server Configuration Guide for J2EE. Also, this feature is only supported on Windows servers.

3.3.3 Multi-thread download settings

Multi-thread downloads boost the application runtime performance. However, if there are many threads competing for the processing power of Web server, it may slow down the performance of Web server. Therefore, you should not specify an unnecessarily large number of threads in AEM. It may take some trial and error to fine-tune the performance.

3.3.4 Custom Libraries download settings

If your application utilizes a customer library (e.g. DLL, OCX, etc.), you can specify in AEM how the custom library should be downloaded to the client:

- In most situations you should set the install mode to "Install automatically without asking user" or "Confirm with user, then install automatically". With these options, the custom libraries will be automatically downloaded and seamlessly installed to the Web browser.

- However, if the file size of the custom libraries is extremely large (e.g. tens of megabytes), you should set the install mode to "Install manually (no automatic installation)".With this option, Appeon does not automatically download and install the custom libraries. As such, you must distribute the custom libraries to users and your users need to install it manually.

For more details about the custom libraries download settings, refer to Section 5.4.4.3, "DLL/OCX Files" in *Appeon Server Configuration Guide for .NET* or in Appeon Server Configuration Guide for J2EE.

3.3.5 Log file settings

Once your application is fully tested and ready to move to a production environment, it is recommended to disable the AEM log functionality. Writing log files incurs disk activity, which can impact performance. Generally, the impact is small but nonetheless it will not hurt to disable this.

3.4 Internet Explorer performance settings

For optimal performance, it is recommended that the Web file caching functionality of Internet Explorer be fully utilized. This will significantly reduce the time required to load and start an application following the initial load. The configuration outlined below will ensure that you realize the best performance while safeguarding your application from becoming "stale".

Step 1: Open Internet Explorer and select **Tools** > **Internet Options**. Verify that the **Empty Temporary Internet Files folder when browser is closed** option is not checked under the **Security** section of the **Advanced** tab of Internet Options.

Step 2: Click the **Settings** button under the **General** tab to configure the Temporary Internet Files settings.

Step 3: Select the **Automatically** radio button and verify that the Amount of disk space to use scroll box is set to a reasonable number, such as 200 MB or more.

Now the browser is set to automatically check for newer versions of the Web application.

3.5 Web and application server performance settings

3.5.1 SAP Sybase EAServer

The core processing of your Web application happens on the application server. As such, the better EAServer performs the better your Web application will perform. This section highlights several key performance settings you should definitely consider. You may refer to the *EAServer Performance and Tuning Guide* for details on how to extensively tune EAServer.

3.5.1.1 JVM startup option

When starting EAServer, the -jvmtype switch specifies whether the client, server, or classic Java VM be used. It is recommended that you set the switch to the J2EE server VM.

3.5.1.2 Configuring data sources

3.5.1.2.1 JDBC driver used by EAServer data source

You should avoid using any JDBC-ODBC driver. Instead, use the Native-protocol/all-Java driver. The only exception is SAP Sybase SQL Anywhere, which provides the iAnywhere JDBC driver that actually performs remarkably well. For detailed information on configuring JDBC drivers refer to Section 4.2.4, "JDBC driver preparation" in *Appeon Server Configuration Guide for J2EE*.

3.5.1.2.2 Configuring the cache size

By default, EAServer is configured to establish 10 database connections. For applications with many several hundred or several thousands of users, this number is often too small. If not increased, it will hurt the performance of your application since your users will be waiting in a queue for an available database connection. On the other hand, if your Web deployment is few hundred users or less, you should not specify an unnecessarily large number as it will consume more server resources and possibly negatively affect performance.

The maximum pool size property, "com.sybase.jaguar.conncache.poolsize.max", defines the maximum number of connections to be held in the connection pool. The size property is generally set to 10%-20% of the maximum number of concurrent users. However, it is possible to use the FORCE option when connecting to obtain additional connections outside of the pool if none are available. Refer to the *EAServer Performance and Tuning Guide* for details on tuning the cache size.

3.5.1.3 HTTP properties

You should configure the EAServer http.maxthreads and server.maxconnections properties such that it can handle the expected concurrent user load for the Web Server. If these properties are improperly configured, it may result in poor performance and possibly result in failed HTTP requests. The httpstat.dat file keeps track of cumulative hits on http objects.

Set the http.maxthreads property to the estimated average number of concurrent HTTP requests (including Servlets and any other server pages). For example, if it is expected that there will be 100 concurrent requests, set the http.maxthreads slightly higher (for example, 120). This will give you a margin of safety. Similarly, set the server.maxconnections to accommodate the average number of concurrent IIOP requests that are expected.

Another property you should pay attention to is server.maxthreads. Set this property to equal the combined value of http.maxthreads and server.maxconnections, and add 50 as a margin of safety (http.maxthreads + server.maxthreads + 50). However, if you are using an older version of PowerBuilder components with a bind thread set, you should increase this number as outlined in the *EAServer Performance Tuning Techniques* document.

Since each application and environment is unique, these are starting points that need to be monitored and adjusted for optimal results. Load and stress testing your application will help you to identify any issues prior to moving your Web application to a production environment.

3.5.2 Microsoft IIS server

Tune the IIS server will help you to avoid some common errors and gain a better performance of Web applications. This section highlights the key performance settings of IIS. You can refer to the IIS Operation Guide for details on how to extensively tune IIS server.

3.5.2.1 Recommendations for avoiding common errors on IIS

IIS should be tuned to avoid the contention, poor performance, and deadlocks for the Web applications. Following are some samples of commons error that you may find in Application log, System log or even log in the Web browser.

• Event Type: Error

  Event Source: ASP.NET

Event ID: 1003

Description: aspnet_wp.exe (PID: <xxx>) was recycled because it was suspected to be in a deadlocked state. It did not send any responses for pending requests in the last 180 seconds.

- Event Type: Warning

  Event Source: W3SVC-WP

  Event ID: 2262

  Description: ISAPI 'C:\Windows\Microsoft.net\Framework\v.2.0.50727\aspnet_isapi.dll' reported itself as unhealthy for the following reason: 'Deadlock detected'.

- Event Type: Warning

  Event Source: W3SVC

  Event ID: 1013

  Description: A process serving application pool 'DefaultAppPool' exceeded time limits during shut down. The process id was '<xxxx>'.

- You may receive the exception error message: "System.InvalidOperationException: There were not enough free threads in the ThreadPool object to complete the operation."

- In the browser, you may also receive the exception error message: "Request timed out".

To avoid such problems, you can configure ASP.NET according to the suggestions below so to best fit your situation and make the Web application perform better. Details on how to configure the ASP.NET can be found at Advanced thread settings.

1. Limit the number of .NET requests that can execute at the same time to approximately 12 per CPU.

2. Permit Web service callbacks to freely use threads in the ThreadPool.

3. Select an appropriate value for the maxconnections parameter. Base your selection on the number of IP addresses and AppDomains that are used.

3.5.2.2 Advanced thread settings

The following settings are the most-Appeon-related settings in ASP.NET.

When you call an Appeon web application from IIS, you may experience contention, poor performance, and deadlocks. Clients may report that requests stop responding (or "hang") or take a very long time to execute.

This problem might occur because ASP.NET limits the number of worker threads and completion port threads that a call can use to execute requests. If there are not sufficient threads available, the request is queued until sufficient threads are free to make the request. Therefore, .Net will not execute more than following number of requests at the same time:

(maxWorkerThreads*number of CPUs)-minFreeThreads

Note: The minFreeThreads parameter and the minLocalRequestFreeThreads parameter are not implicitly multiplied by the number of CPUs.

3.5.2.2.1 maxWorkerThreads and maxIoThreads

.NET uses the following two configuration settings to limit the maximum number of worker threads and completion threads that are used:

```
<processModel maxWorkerThreads="20" maxIoThreads="20" />
```

The maxWorkerThreads parameter and the maxIoThreads parameter are implicitly multiplied by the number of CPUs, the default value of these two parameters are both 20. If for some reason your application is slow, perhaps waiting for external resources, you could try to increase the number of threads to a value less than 100. For example, if you have two processors, the maximum number of worker threads is the following: 2*maxWorkerThreads

3.5.2.2.2 minWorkerThreads

The setting determines how many worker threads may be made available immediately to service a remote request. By default, the minWorkerThreads parameter is not present in either the Web.config file or the Machine.config file at C:\WINDOWS\Microsoft.NET\Framework \v2.0.50727\CONFIG. You need to manually add the following line to make the setting work.

```
<processModel minWorkerThreads="1" />
```

Threads that are controlled by this setting can be created at a much faster rate than worker threads that are created in other ways. The default value for the minWorkerThreads parameter is 1. The setting is recommended to set in the following way.

minWorkerThreads = maxWorkerThreads / 2

Note: This setting is implicitly multiplied by the number of CPUs.

3.5.2.2.3 minFreeThreads and minLocalRequestFreeThreads

The two settings determine how many worker threads and completion port threads must be available to start a remote request or a local request:

```
<httpRuntime minFreeThreads="8" minLocalRequestFreeThreads="8" />
```

The default value is 8. If there are not sufficient threads available, the request is queued until sufficient threads are free to make the request. Therefore, .NET will not execute more than the following number of requests at the same time:

(maxWorkerThreads*number of CPUs)-minFreeThreads

Note: The minFreeThreads parameter and the minLocalRequestFreeThreads parameter are not implicitly multiplied by the number of CPUs.

3.5.2.2.4 maxconnection

The maxconnection parameter determines how many connections can be made to a specific IP address. The parameter appears as follows:

```
<connectionManagement>
  <add address="*" maxconnection="2" />
  <add address="65.53.32.230" maxconnection="12" />
</connectionManagement>
```

The maxconnection parameter setting applies to the AppDomain level. By default, because this setting applies to the AppDomain level, you can create a maximum of two connections to a specific IP address from each AppDomain in your process.

3.5.2.2.5 execution Timeout

The setting limits the request execution time:

```
<httpRuntime executionTimeout="90" />
```

The default is 110 seconds.

Note: If you increase the value of the executionTimeout parameter, you may also have to modify the processModel responseDeadlockInterval parameter setting.

3.5.2.2.6 Recommended thread settings

For most applications, you can use and apply the recommended changes in the Machine.config file as below, which can be found at C:\WINDOWS\Microsoft.NET \Framework\v2.0.********\CONFIG:

1. Set the values of the maxWorkerThreads parameter and the maxIoThreads parameter to 100.

2. Set the value of the maxconnection parameter to 12*N (N is the number of CPUs that you have).

3. Set the values of the minFreeThreads parameter to 22*N and the minLocalRequestFreeThreads parameter to 19*N.

4. Set the value of minWorkerThreads to 50. Remember, minWorkerThreads is not in the configuration file by default. You must add it.

If you have hyperthreading enabled, you must use the number of logical CPUs instead of the number of physical CPUs.

Note: If you have a server with one processor, when you use this configuration, you can execute a maximum of 78 .NET requests at the same time because 100-22=78. Therefore, at least 22*N worker threads and 22*N completion port threads are available for other uses (such as for the Web service callbacks).

For example, if you have a server with four processors and hyperthreading enabled, then n=8 (=2*4). Based on these formulas, you would use the following values for the configuration settings that are mentioned in this section.

```
<system.web>
......
<processModel maxWorkerThreads="100" maxIoThreads="100" minWorkerThreads="50" />
<httpRuntime minFreeThreads="176" minLocalRequestFreeThreads="152" />
<connectionManagement>
  <add address="[ProvideIPHere]" maxconnection="96" />
</connectionManagement>
......
</system.web>
```

Also, if you use this configuration, 12 connections are available per CPU per IP address for each AppDomain, and you can execute a maximum of 624 .NET requests at the same time because 8* 100 - 176= 624.

3.6 Database performance settings

3.6.1 Recommended database driver

The following database drivers are recommended for Appeon for PowerBuilder .NET:

Table 3.2: Recommended database drivers

Database Type	Recommended Driver
Sybase ASA/SQL Anywhere	Native Driver
ASE	ODBC Driver
SQL Server	Native Driver
Oracle	Native Driver
Informix	Native Driver
Teradata	Native Driver

3.6.2 Recommended database setting

Setting appropriate values for the database parameters based on the actual needs can reduce the occurrence of database deadlock and block hence can improve the concurrency and stability of the Web application.

3.6.2.1 Command Timeout

Specify the timeout period for the commands. In the following table, it shows how to set Command Timeout to 120 seconds in different databases.

Table 3.3: Command timeout settings in different databases

Database Type	Parameter	Remark
Sybase ASA/SQL Anywhere	BLOCKING_TIMEOUT	Execute the following command: SET OPTION Public.BLOCKING_TIMEOUT = 120000
ASE	lock wait period	Execute the following command: sp_configure "lock wait period", 120
SQL Server	-	Modify the Command Timeout parameter in Edit Data Source page of AEM. Command Timeout = 120
Oracle	IDLE_TIME	Execute the following command: ALTER PROFILE default LIMIT IDLE_TIME 2;

Database Type	Parameter	Remark
Informix	DEADLOCK_TIMEOUT	Modify the DEADLOCK_TIMEOUT parameter in ONCONFIG.std file. DEADLOCK_TIMEOUT = 120
Teradata	-	Modify the Command Timeout parameter in Edit Data Source page of AEM. Command Timeout = 120

4 Identifying Performance Bottlenecks

4.1 Overview

There are several methods to identify performance bottlenecks in your application. You can manually test your application or utilize Appeon's built-in performance reporting tool. Manually testing is the most time-consuming but also the most comprehensive and accurate. Nonetheless, the performance reporting tool, can help to identify most problematic Windows without a lot of work.

4.2 Appeon Performance Analyzer

Appeon Performance Analyzer is a tool in Appeon Server Web Component to help diagnose application performance by analyzing script calls, application RPC calls, application errors and application downloads, and thereby calculate the time used by function calls or object calls.

4.2.1 Getting Started

Before we start to work with Appeon Performance Analyzer, let's learn how to start it and what it looks like first.

- [Enabling Appeon Performance Analyzer](#)

- [Starting Appeon Performance Analyzer](#)

- [Getting to Know Appeon Performance Analyzer](#)

- [Removing Appeon Performance Analyzer](#)

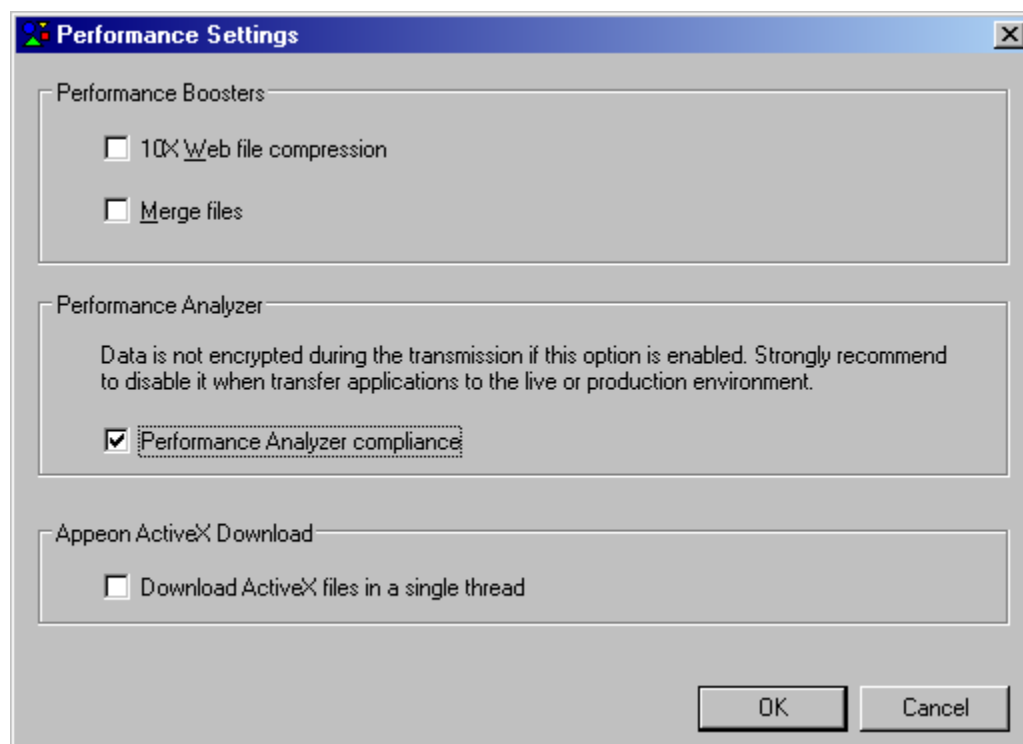#### 4.2.1.1 Enabling Appeon Performance Analyzer

Appeon Performance Analyzer is turned on or off at application level. If you want to analyze an application by using Appeon Performance Analyzer, you need to deploy the application in a mode compliant with Performance Analyzer. To do this:

Step 1: Select the *Performance Analyze compliance* option in the Application Profile Configuration tool in Appeon Developer. It is not selected by default.

Step 2: Deploy the application using the Appeon Deployment Wizard in Appeon Developer.

The application files are deployed and generated in a mode compliant with the Performance Analyzer tool.

Note: as the data is not encrypted during transmission at runtime, therefore, the *Performance Analyze compliance* option is recommended to use only at the test or development stage. When the application is ready to go live in a production environment, turn off this option and then re-deploy the application, to ensure data is encrypted during transmission.

Figure 4.1: Performance Analyze compliance option

4.2.1.2 Starting Appeon Performance Analyzer

Performance Analyzer is a Web tool that can only be launched side by side with the Web application you want to analyze.

Notes:

1. Appeon Performance Analyzer can only run on a 32-bit operating system.

2. For the first time to run it, users should log into Windows as an administrator and turn off UAC (if using Windows Vista, Windows 7 etc.). To turn off UAC, go to Windows **Control Panel** > **User Accounts**, click **Turn User Account Control on or off**, then uncheck the check box of **Use User Account Control (UAC) to help protect your computer**.

You can start Appeon Performance Analyzer by the following steps:

Step 1: Run the Web application you want to analyze. The URL of the Web application should look like this: http://hostname:port/applicationname/x32_index.htm.

Note: Hostname indicates the IP address or machine name of the Web Server, and port indicates the HTTP port number of the Web Server. Port is optional and specifies the port number that your Web Server accepts for HTTP connections. If it is not specified, the default port is 80. If several Web Servers are running on the same machine, make sure that the port specified is a port of the Appeon Web Server.

Step 2: When the application has started up, append **"?performance=1"** to the application URL in the address bar and press Enter to refresh the application.

So the complete URL should look like http://hostname:port/applicationname/x32_index.htm**?performance=1** orhttp://localhost/pet_world/x32_application.htm**?performance=1**.

Then the main window of the Performance Analyzer pops up.

Note: A prompt window pops up if the Performance Analyzer is run for the first time.

Figure 4.2: Performance analyze prompt
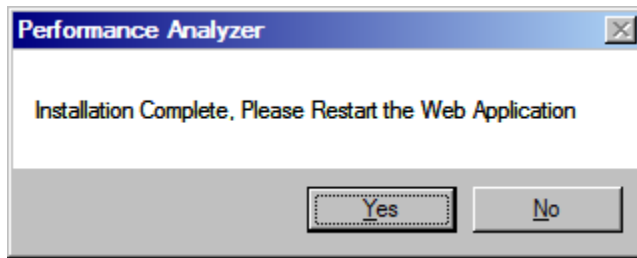
Click **Yes** to close the browser running the Web application; then re-do Step 1 and 2; Appeon Performance Analyzer will start up side by side with the Web application. Or click **No**, to let the Web application continue to run as it is.

The main window of Appeon Performance Analyzer looks like this.

Figure 4.3: Performance analyzer main window
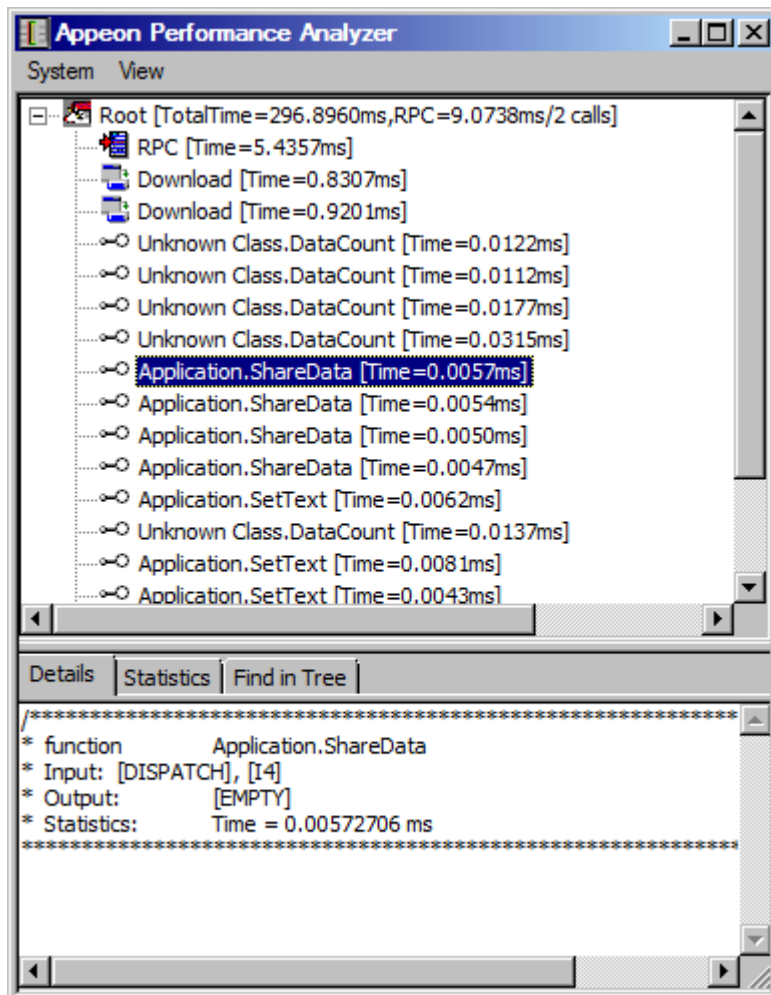
Appeon Performance Analyzer will automatically open when the Web application runs by URL http://hostname:port/applicationname/x32_index.htm?performance=1, unless you deselect the **Show window when started** option in the **Appeon Performance Analyzer Configuration** dialog (from menu **System** > **Config**). See System Configuration for details.

4.2.1.3 Getting to know Appeon Performance Analyzer

Appeon Performance Analyzer window includes three parts:

1 - The menu bar

The menu bar includes all the settings for Performance Analyzer. You can get detailed information from the sections below.
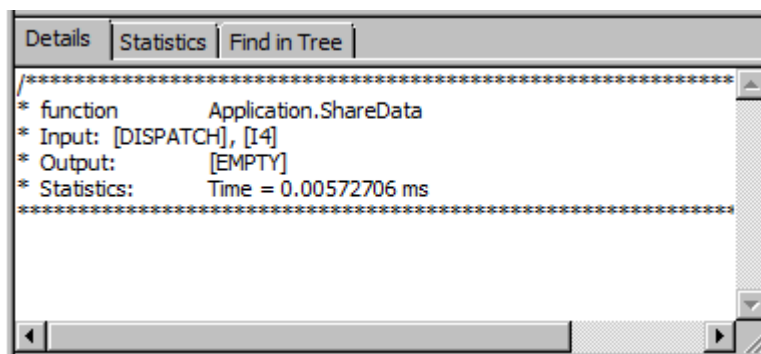
2 - Analyzing panel

The Analyzing panel displays the data analyzed by the Performance Analyzer in a treeview. Each node of the tree represents a JS call, AX call, RPC call or Download.

Figure 4.4: Analyzing panel
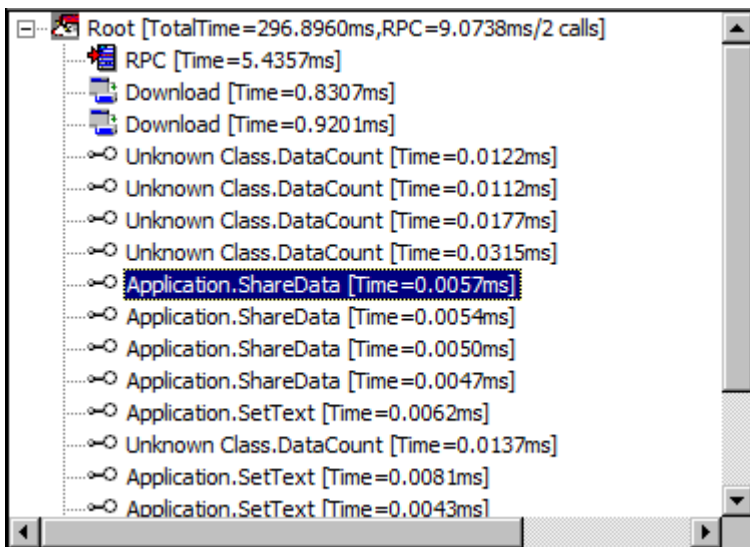
3 - Detail view panel

Figure 4.5: Detail view panel

The Detail view panel shows the detailed information of the selected tree node, and also shows the statistics and search results.

4.2.1.4 Removing Appeon Performance Analyzer

In order to remove Appeon Performance Analyzer completely, you can follow the steps below:

Step 1: Make sure that the user has administrator permission, and then turn off UAC and IE.

Step 2: Open the command window and enter the following commands one by one:

```
regsvr32 -u "%systemroot%\downloaded program files\APBPanalysis.dll"
```

```
del "%systemroot%\downloaded program files\APBPanalysis.dll"
```

```
del "%systemroot%\downloaded program files\APBPanalysis.inf"
```

```
del "c:\performance\EonTracer.ini"
```

```
del "c:\performance\EonTracer.log"
```

```
rd "c:\performance"
```

After all the commands above are run, the Appeon Performance Analyzer is removed completely.

4.2.2 Working with Appeon Performance Analyzer

This section will talk about the following topics:

- System Configuration

- Calls Analysis

- Download Analysis

- View Detail

- Additional functions

4.2.2.1 System Configuration

In order to make Appeon Performance Analyzer more convenient for future use, you can make some configurations in the System menu.

Click menu System | Config. The Appeon Performance Analyzer Configuration dialog opens:

Figure 4.6: Appeon Performance Analyzer Configuration dialog
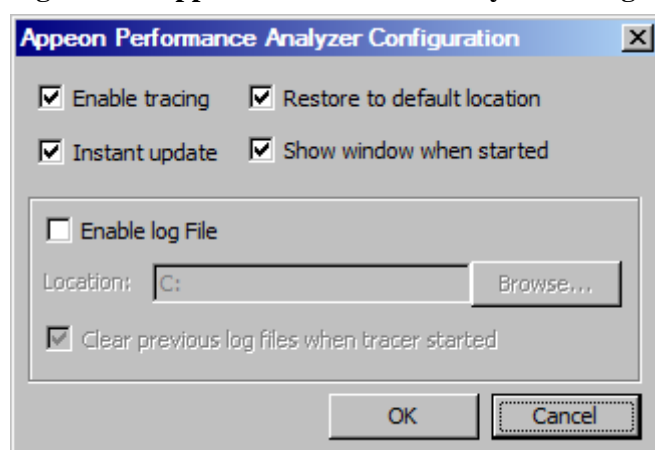
Enable Tracing

This option controls if Appeon Performance Analyzer works or not. If you want to work with Appeon Performance Analyzer to analyze an application program, the option must be selected. By default it is selected.

Instant update

By selecting Instant update, the analyzing data in the analyzing panel can be updated instantly when operation changes. When Instant update is not selected, Performance Analyzer will work faster; but users will need to manually click menu View | Update to see the latest data.

Restore to default location

By selecting this option, the main window of Appeon Performance Analyzer always opens at the default position. By unselecting this option, Appeon Performance Analyzer will remember its last position on the screen when it exits, and will open at the last location on the screen when it starts next time. By default, this option is selected.

Show window when started

By selecting this option, Appeon Performance Analyzer will automatically start when the Web application runs. You can deselect this option to disable the automatic starting of Appeon Performance Analyzer, and manually Start Appeon Performance Analyzer by pressing Ctrl + Shift + F12. By default it is selected.

Enable log file

If you want to save the analyzing log in the local machine, select Enable log file. And the log file will be created after the Web application is closed.

Figure 4.7: Enable log file option
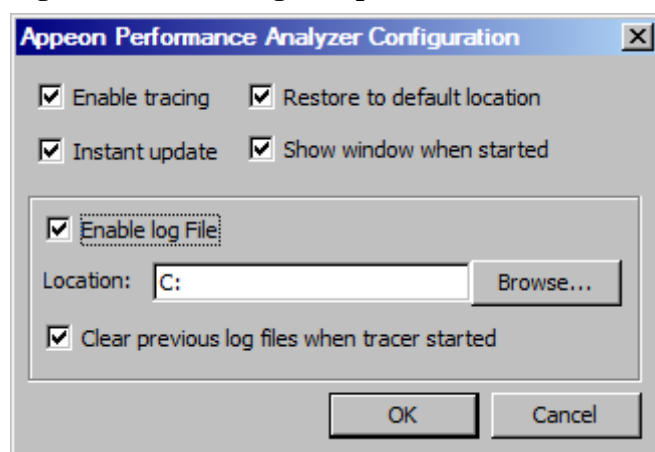
As shown above, the default destination is in C:\ disk. You can specify a different location by clicking *Browse* to select an existing folder on your computer.

Clear previous log files when tracer started

This option becomes selectable when Enable log file is selected. If you want to clear the analyzing log files saved in the last starting of Appeon Performance Analyzer, select this option.

After you finish the configuration, click *OK* to close the dialog.

4.2.2.2 Calls Analysis

Appeon Performance Analyzer enables you to perform the following analysis easily:
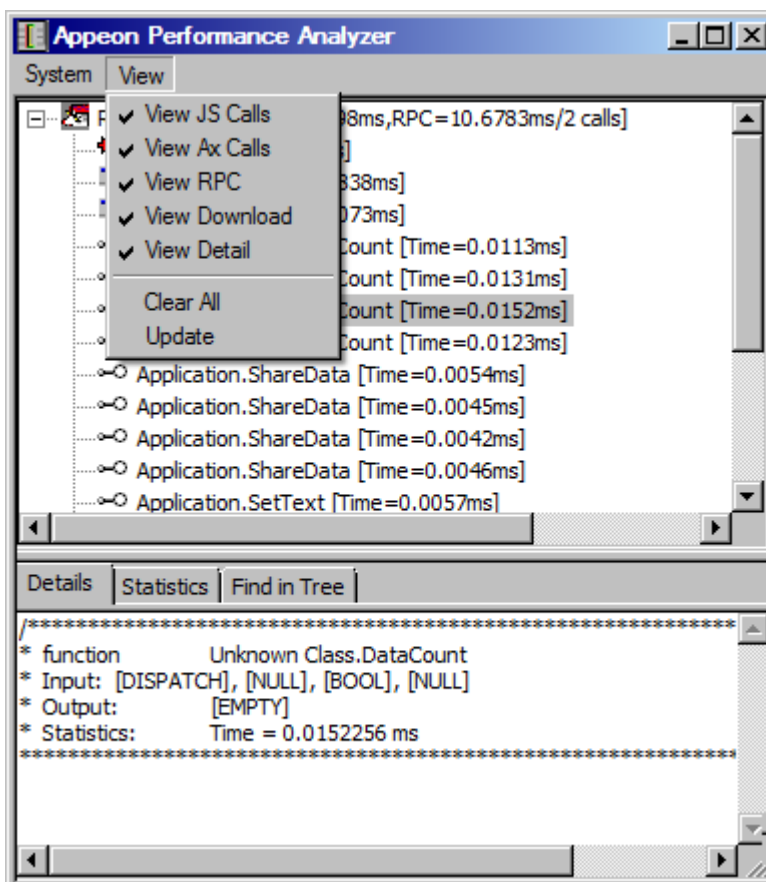
- JavaScript Calls Analysis

- ActiveX Calls Analysis

- Remote Procedure Calls Analysis

No matter what operation you execute on the Web application, e.g. a click on a button to open a dialog in the Web application, a root item Root will show in the first line of the analyzing panel to show the total executing time consumed by the operation.

By expanding Root, you can view the time consumed by different callings.

By default, the JS Calls, Ax Calls, RPC, Download, Detail View buttons are selected, which means, information related with them will be automatically displayed in the analyzing panel and the detail view panel.
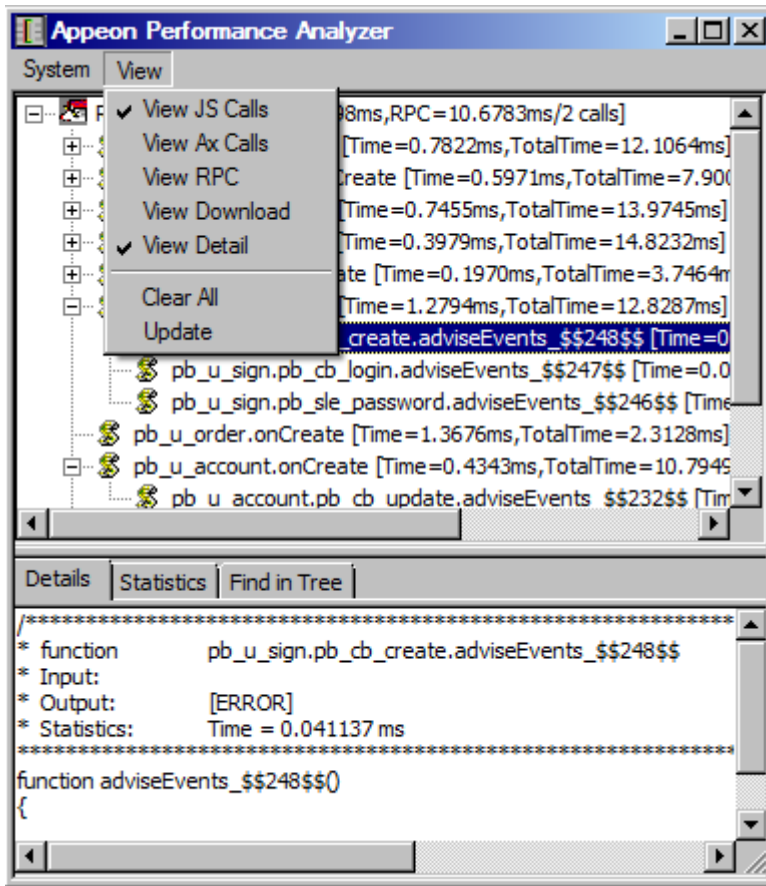
Figure 4.8: View menu

4.2.2.2.1 JavaScript Calls Analysis

By default, the View | View JS Calls menu item is selected, so the JavaScript calls are automatically listed in the analyzing panel. If you do not want to view the JavaScript calls in the analyzing panel, you can click *View JS Calls* menu item to unselect it. If you want to view the JavaScript calls only, you can click the other menu items in the View menu to unselect them, as shown in the figure below.

You can expand the sub-nodes in the JavaScript calls list to further find out the time consumed by each JavaScript function calls.

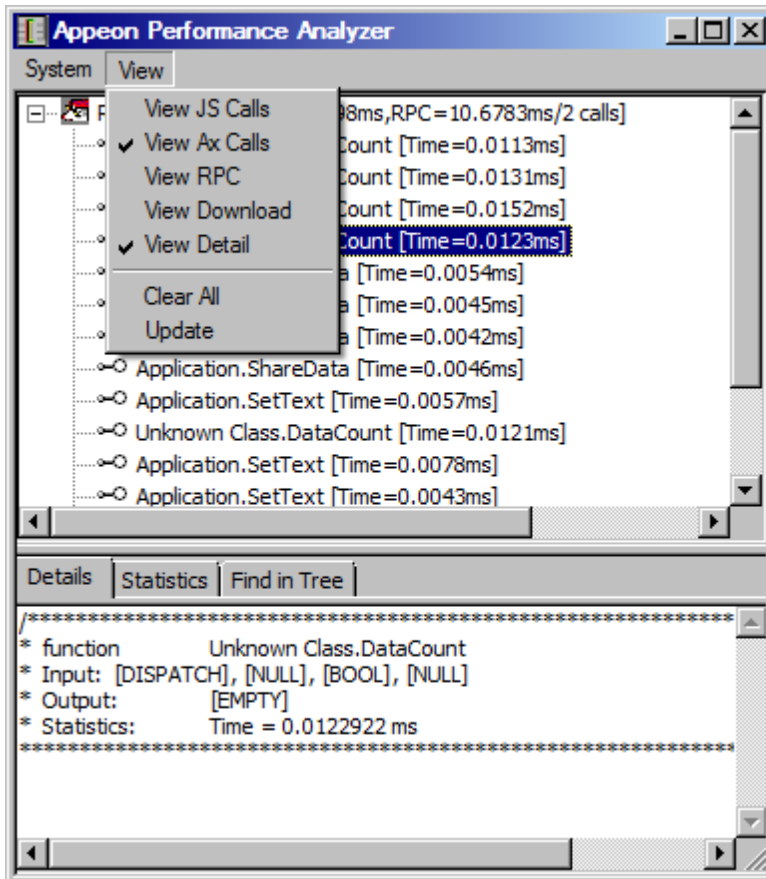You can see more details of JavaScript calls in the detail view panel.

Figure 4.9: View JS Calls menu

4.2.2.2.2 ActiveX Calls Analysis

By default, View AX Calls menu item in the View menu is selected, so the ActiveX calls are automatically listed in the analyzing panel. If you do not want to view the ActiveX calls in the analyzing panel, you can click *View AX Calls* menu item to unselect it. If you want to view the ActiveX calls only, you can click the other optional menu items in the View menu to unselect them, as shown in the figure below.

You can see more details of the ActiveX calls in the Detail View panel.

Figure 4.10: View AX Calls menu

4.2.2.2.3 Remote Procedure Calls Analysis

By default, View RPC in the View menu is selected, so the RPC calls are automatically listed in the analyzing panel. If you do not want to view the RPC calls in the analyzing panel, you can click *View RPC* menu item to unselect it. If you want to view the RPC calls only, you can click the other optional menu items in the View menu to unselect them, as shown in the figure below.

You can see more details of the RPC calls in the Detail View panel.

Figure 4.11: View RPC menu
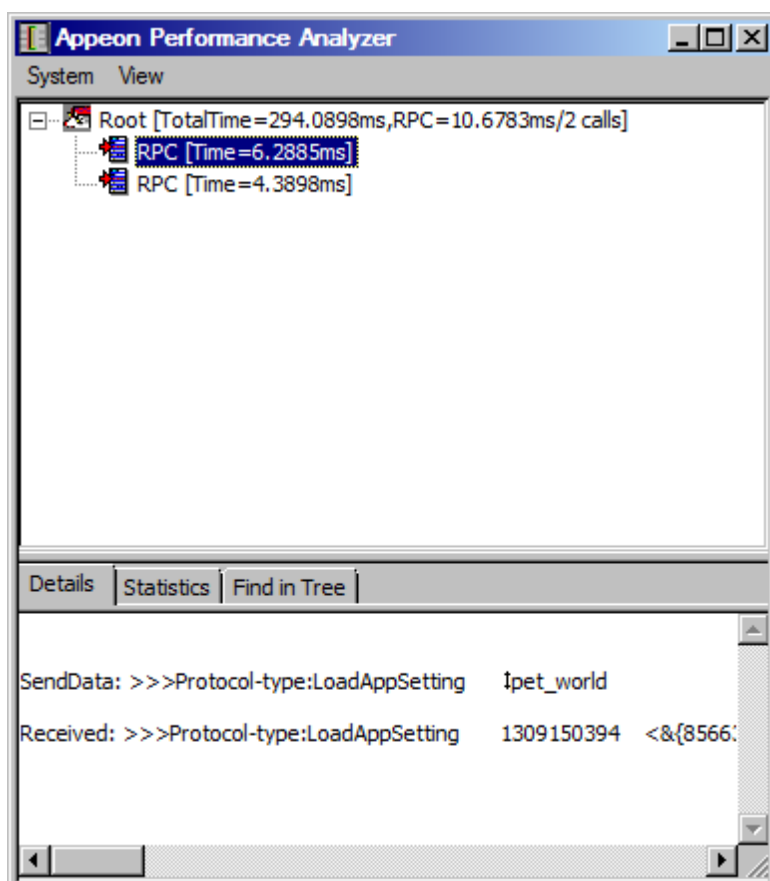
4.2.2.3 Download Analysis

By default, View Download in the View menu is selected, so the downloads are automatically listed in the analyzing panel. If you do not want to view the downloads in the analyzing panel, you can click the *View Download* menu item to unselect it. If you want to view the downloads only, you can click the other optional menu items in the View menu to unselect them, as shown in the figure below.

You can see more details of downloads in the Detail View panel.

Figure 4.12: View Download menu
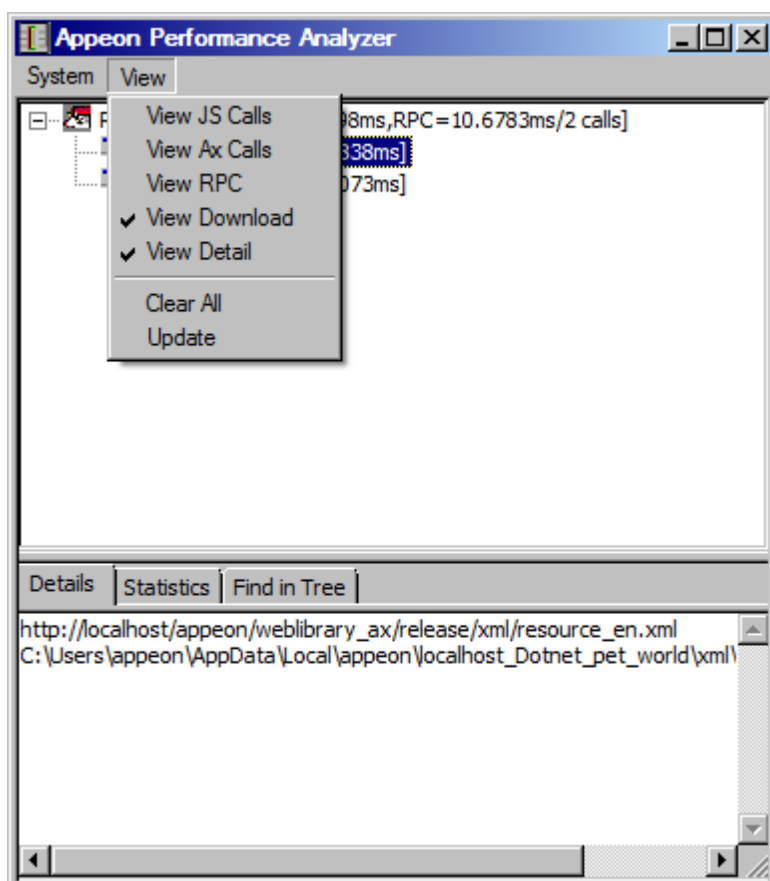
4.2.2.4 View Detail

By default, the View | View Detail menu item is selected, so the View Detail panel is automatically displayed beneath the analyzing panel. If you do not want to see the View Detail panel, you can click the *View Detail* menu item to unselect it.

The detail view panel enables you to view the details and statistics of calls.

View details of JavaScript calls in the detail view panel:

Display the View Detail panel by clicking the *View Detail* menu item in the View menu.

Click the *View JS Calls* menu item in the View menu to display the JavaScript calls in the analyzing panel, then select a JavaScript call, the details of the selected call are displayed in the Details tab of the detail view panel.

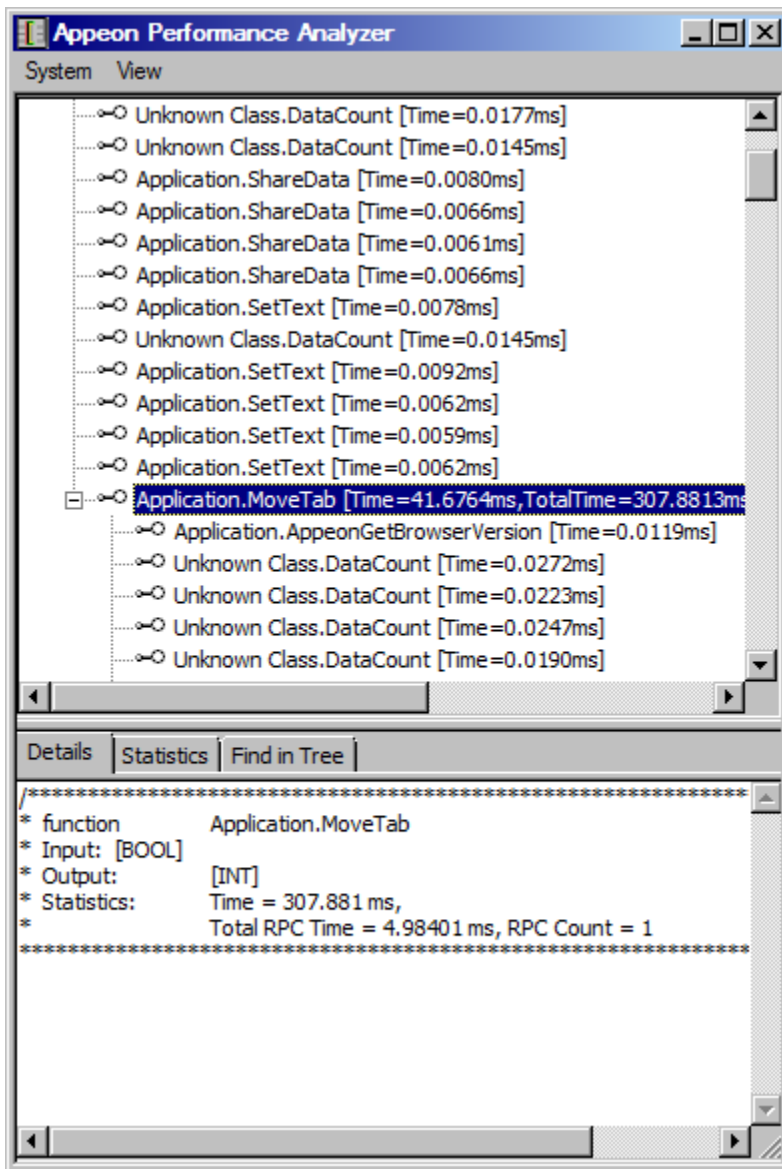**Figure 4.13: View details of JavaScript calls**
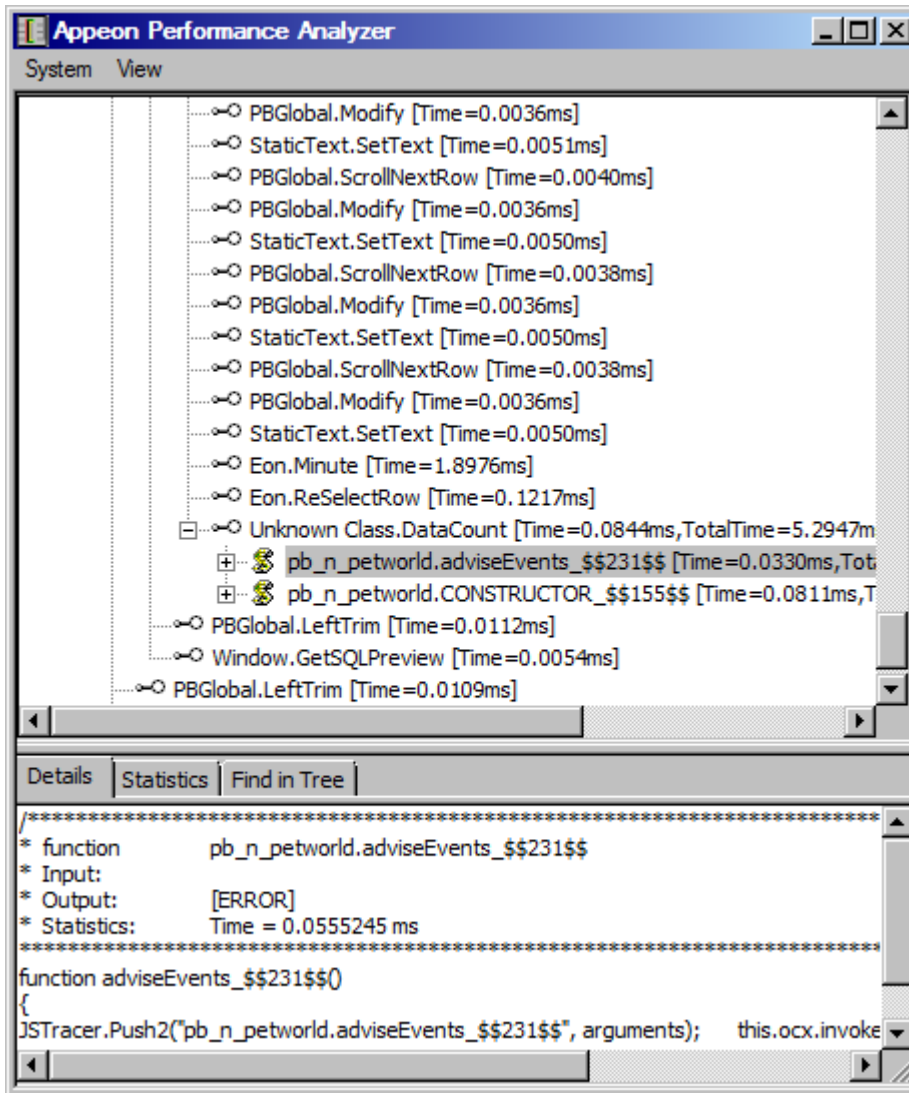
In the Statistics tab of the detail view panel, it shows the calls statistics as below:

Figure 4.14: View statistics of JavaScript calls
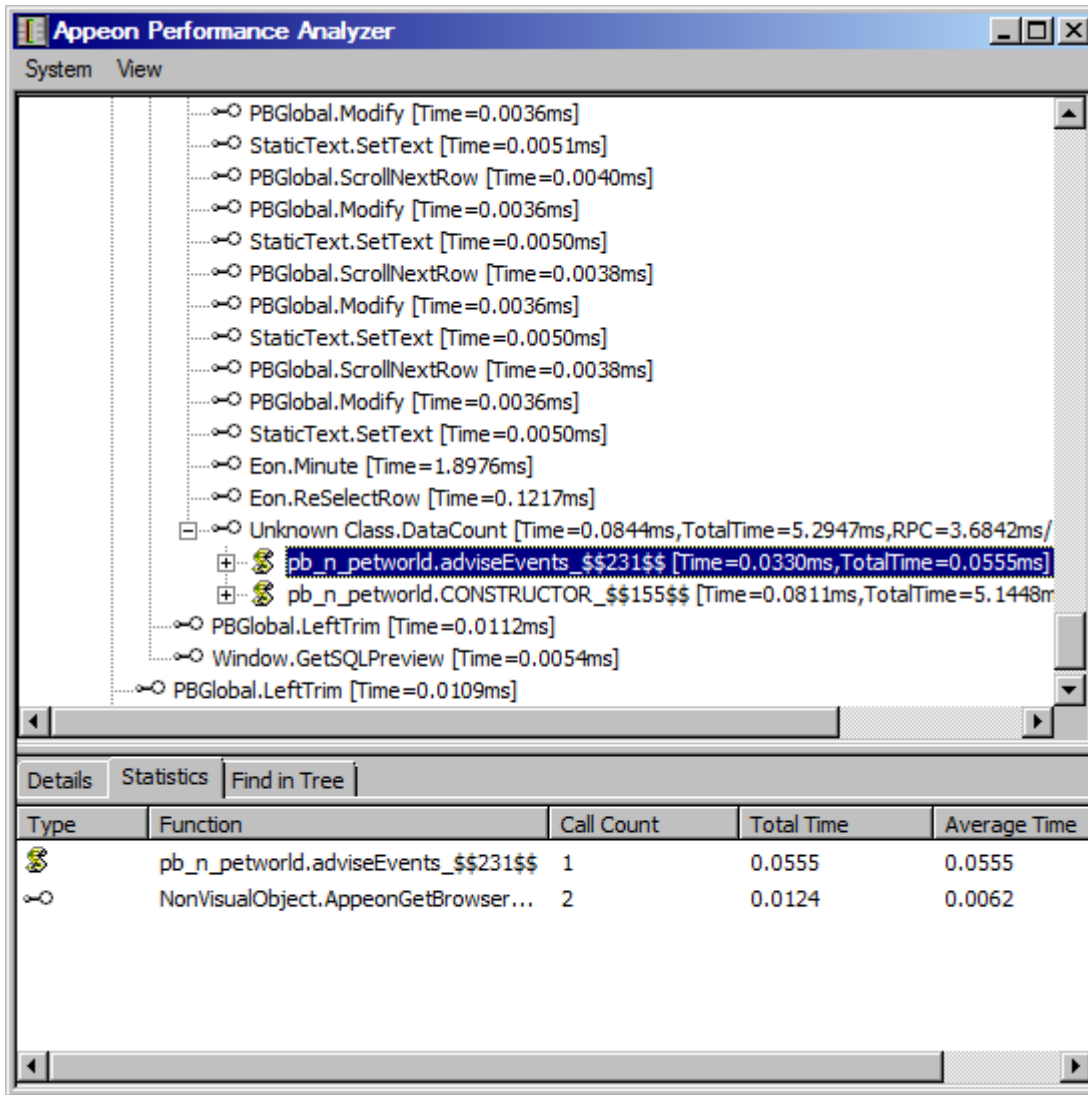
View details of ActiveX calls in the detail view panel in a similar way as JavaScript calls:

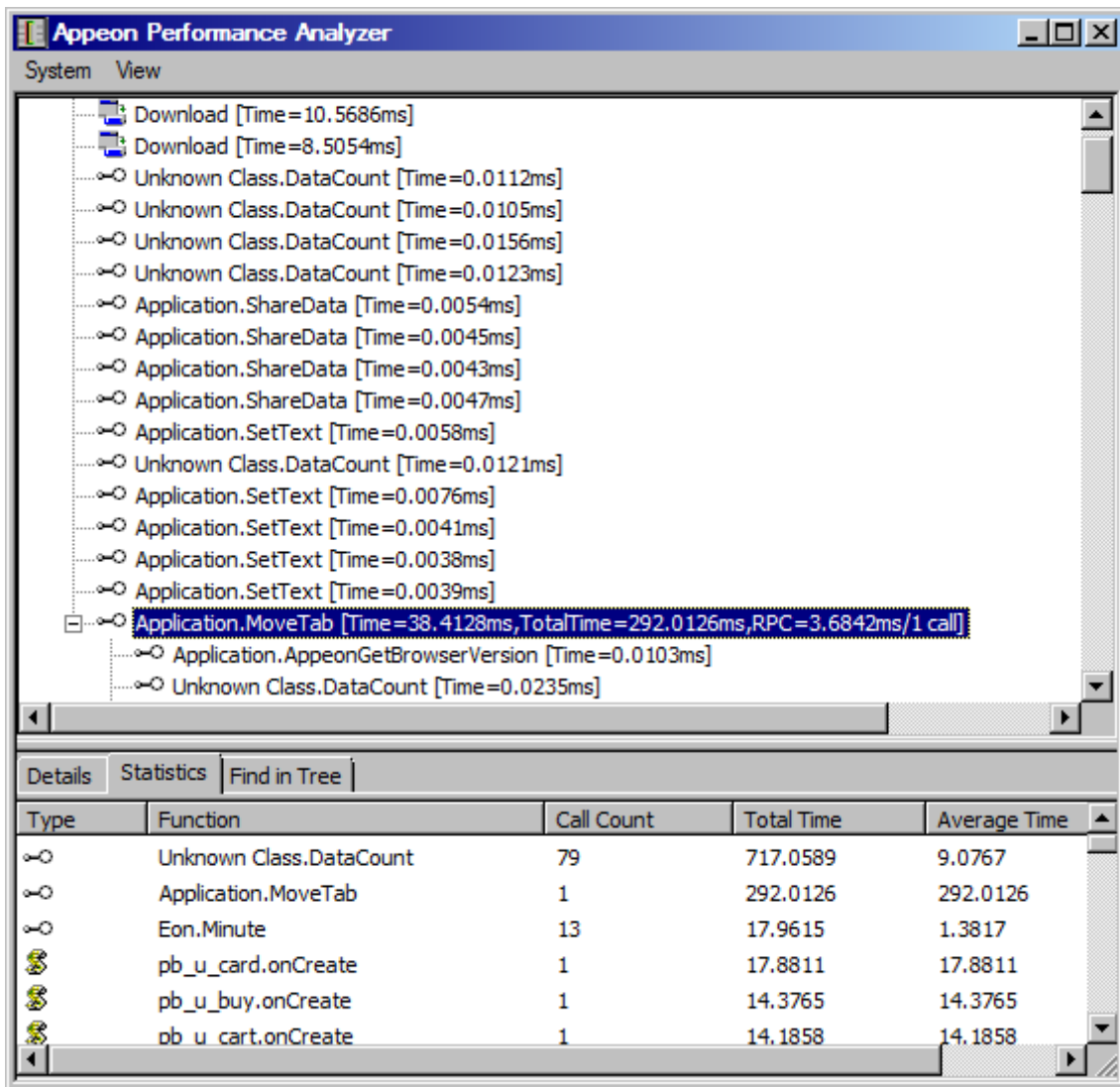**Figure 4.15: View details of ActiveX calls**

View statistics of ActiveX calls in the detail view panel in a similar way as JavaScript calls:

Figure 4.16: View statistics of ActiveX calls

View details of RPC calls in the detail view panel in a similar way as JavaScript calls:

The sending and receiving procedure is roughly displayed in the Details tab, to help you check the RPC process.

Figure 4.17: View details of RPC calls
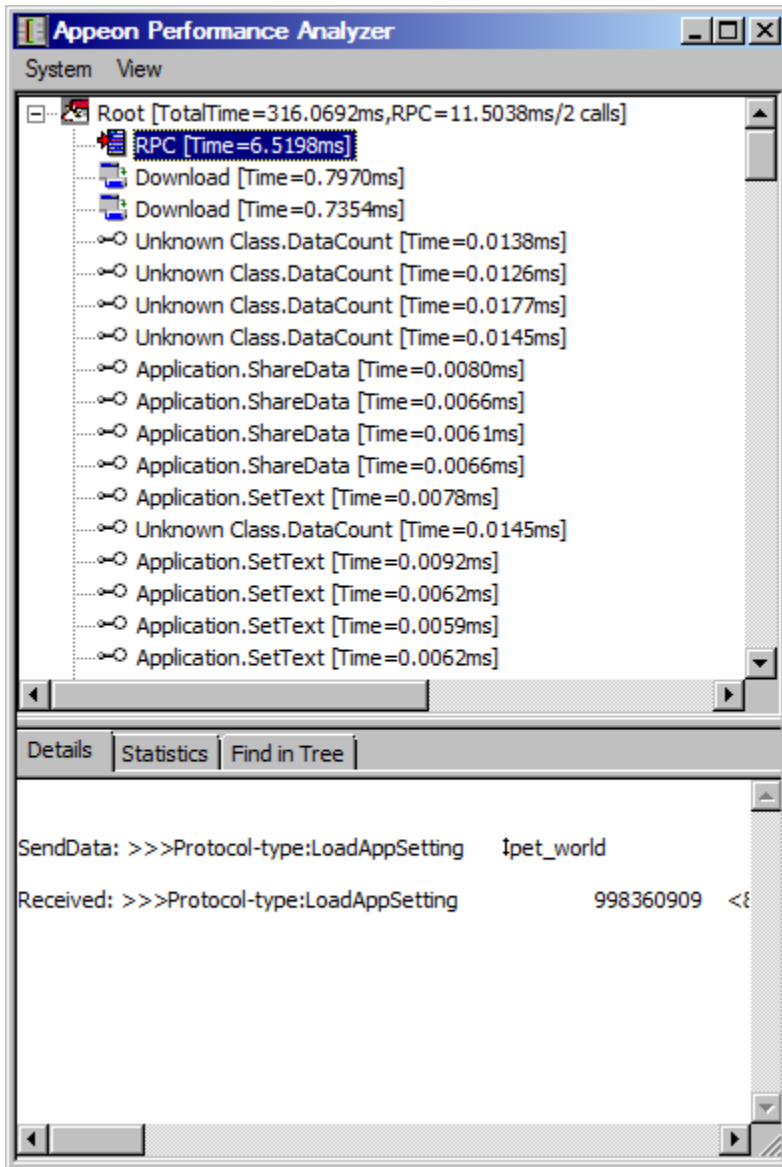
View details of downloads in the Detail View panel:

The Details tab shows the download URL on the Web server and the storage location of the downloaded file.

Figure 4.18: View details of downloads
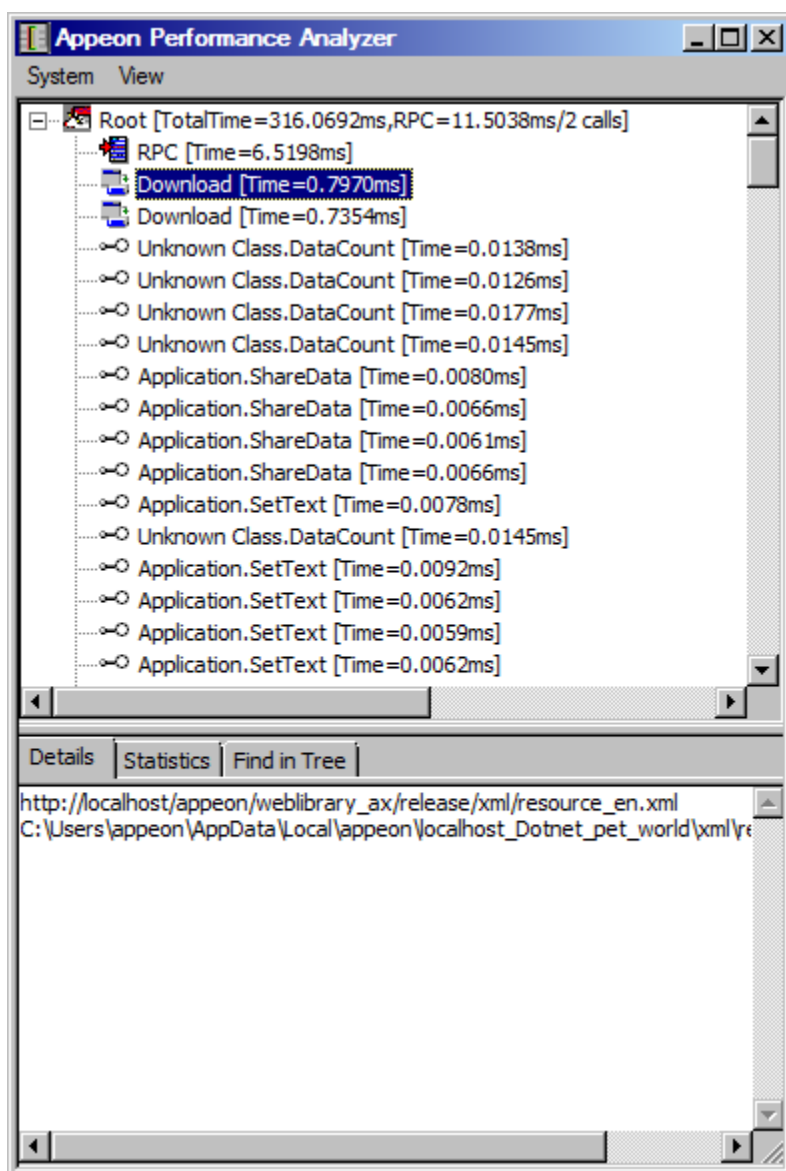
View statistics of root items in the detail view panel:

When you click the item of Root, the Statistics tab shows the statistics for all the calls.

Figure 4.19: View statistics of root items

Apart from the above, the Detail View panel can also show the search results. Refer to the Find function for details.

4.2.2.5 Additional Functions

Appeon Performance Analyzer also provides the following additional functions:
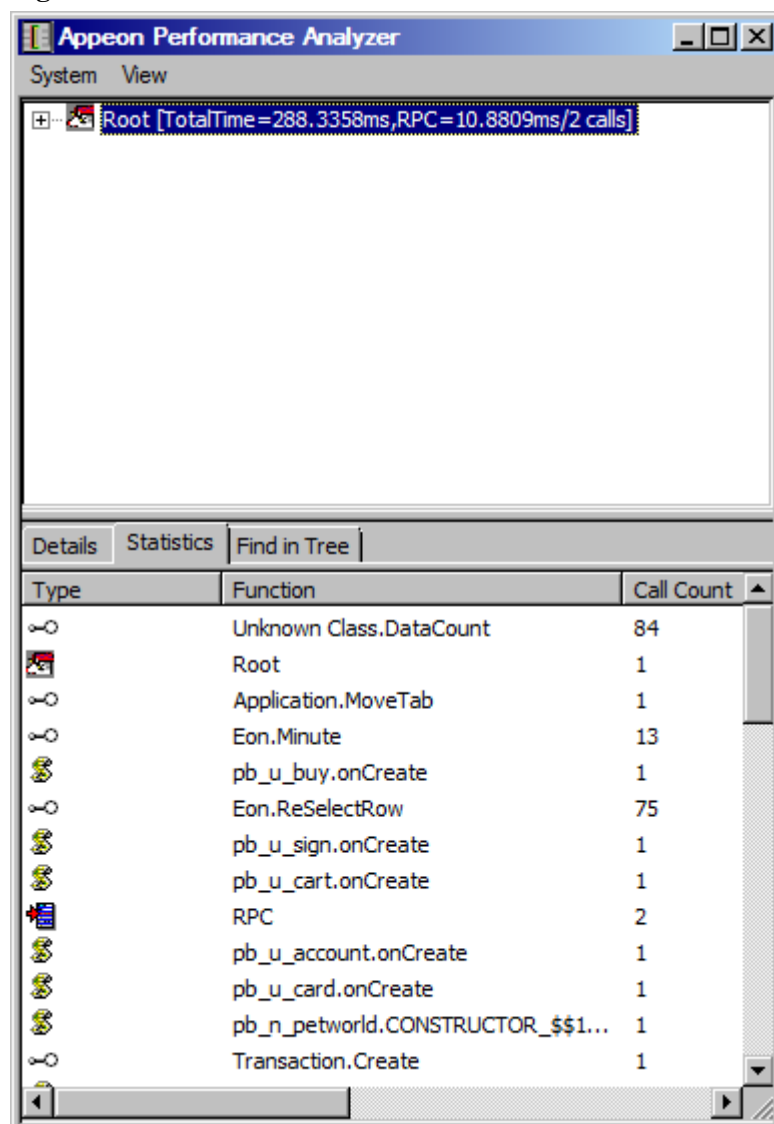
- Filter function

- Find function

- Save As XML function

- New Root Item function

4.2.2.5.1 Filter function

Appeon Performance Analyzer enables you to filter the functions or objects which you do not want to focus on temporarily. Here, in Appeon Performance Analyzer, to filter means to remove.

The filter function is only applicable to ActiveX calling analysis.

Click System | Filter menu item and the Filter Configuration dialog opens. Select Enable Filter.

Note: To enable the Filter function, you must make sure Enable Tracing is selected in System Configuration dialog.

If you want to analyze all the functions or objects in the application, make sure that Enable Filter is not selected.

Figure 4.20: Filter Configuration dialog
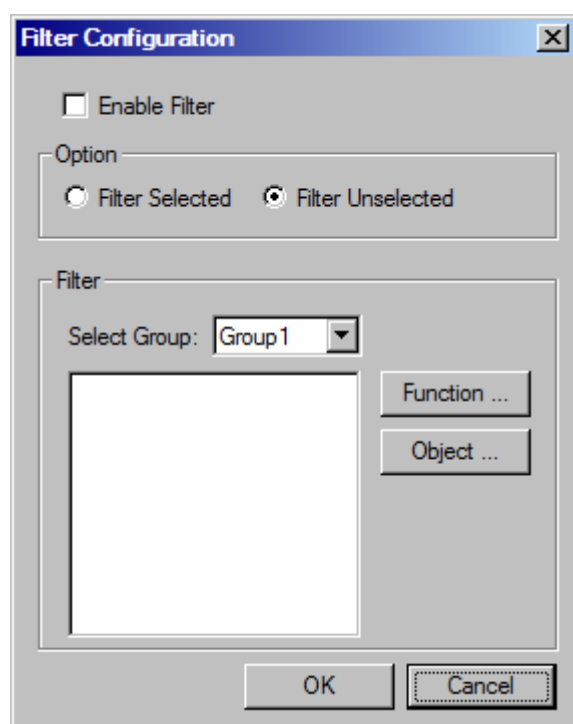
Filter function for ActiveX calls analysis
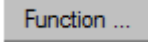
Go through the steps below to learn how to filter objects and functions for ActiveX calls analysis.

Step 1: Select View AX Calls under the View menu, all the ActiveX functions and objects called in the operation are listed.

Figure 4.21: View ActiveX calls
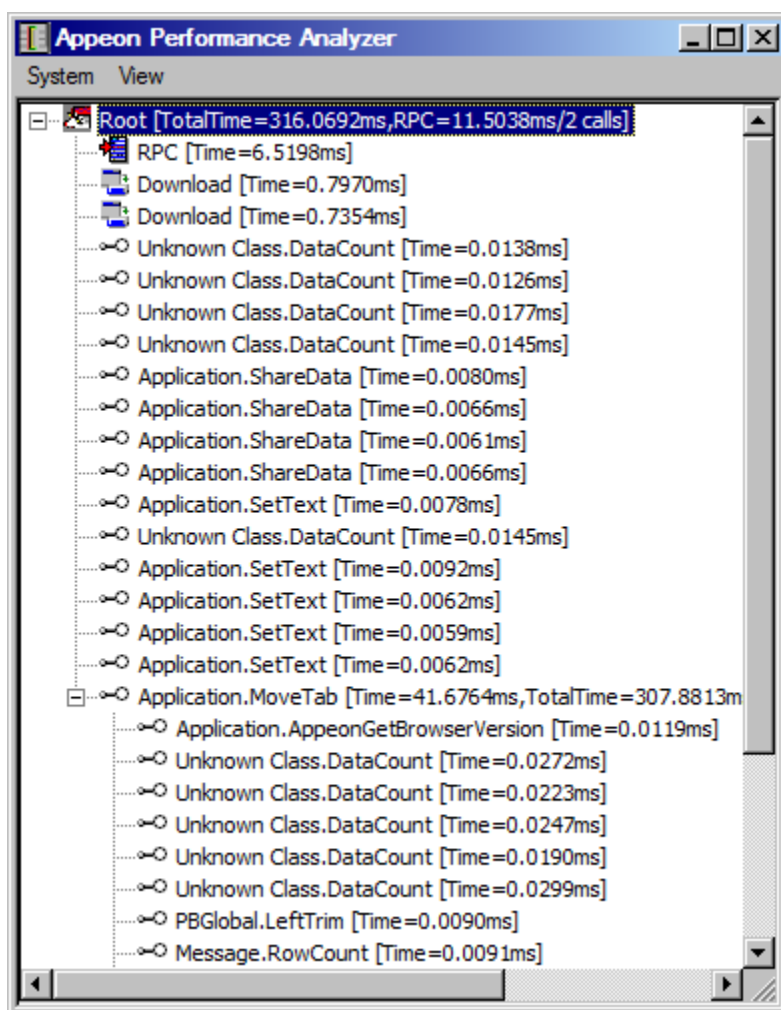
Step 2: Click menu System | Filter. In the Filter Configuration dialog, select Enable Filter.

Figure 4.22: Filter Configuration dialog
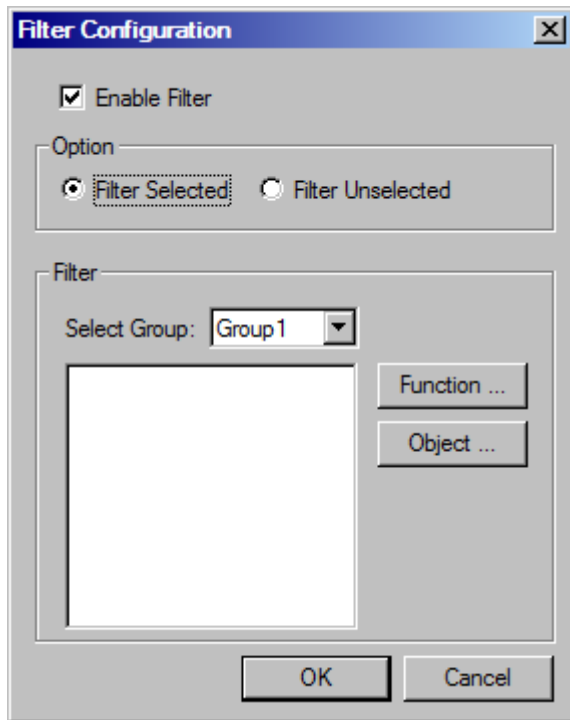
Step 3: In Filter, click the [Function ...] button. The Function Picker dialog opens.

Figure 4.23: Function Picker dialog
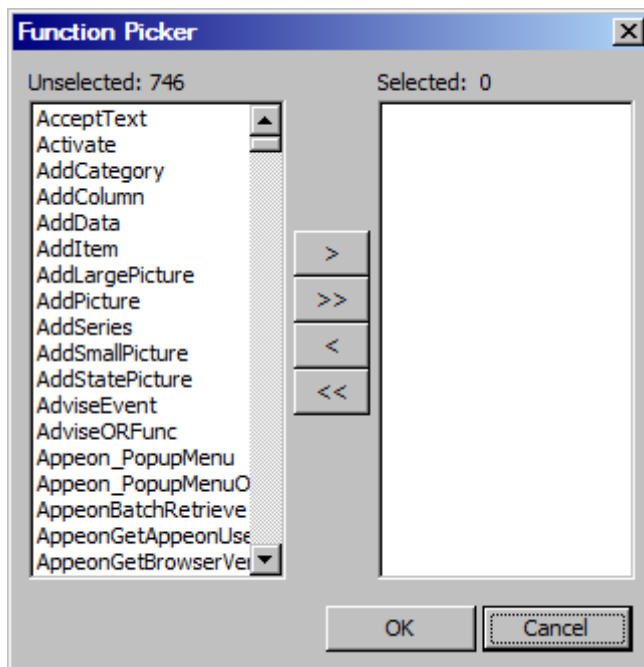
Step 4: Select the functions and/or objects in the Unselected list. Click the arrow button

[>] to move the selected items to the Selected list. Click *OK* to close the dialog.

The double-arrow buttons [>>] and [<<] are used to move all items from one list to the other.

Figure 4.24: Selected functions and objects
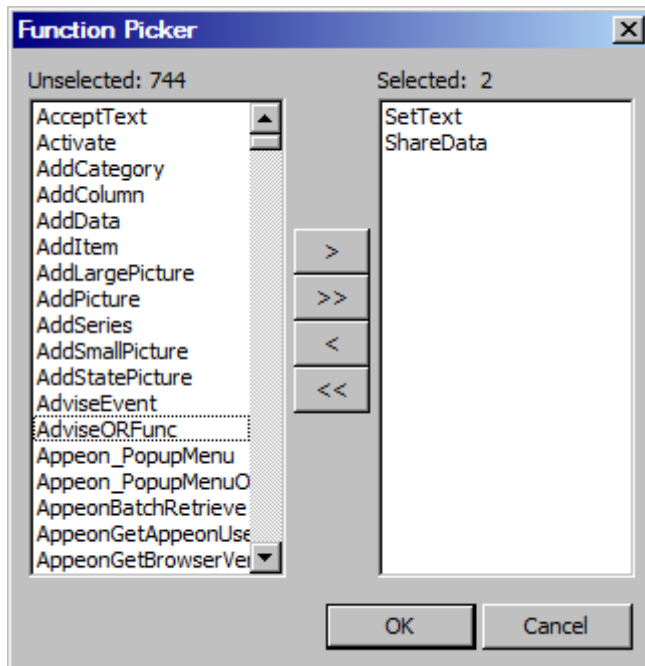
Now, the selected items are in the Filter area.

Note that they are in Group 1. You can have different groups for storing different selected items. The different groups can be all functions, all objects, or mixed with functions and objects.

Figure 4.25: Filter Configuration dialog
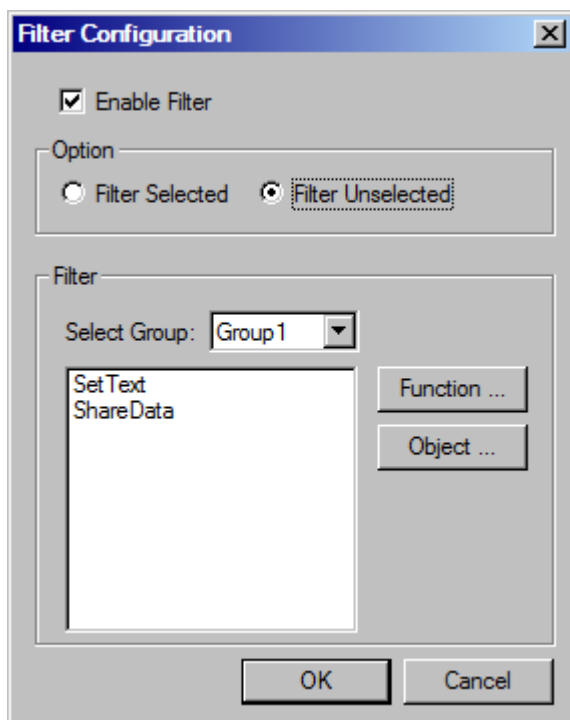
Step 5: In Option area, select Filter Unselected. This will display the selected items only, and hide the unselected items. If you select Filter Selected, it will hide the selected items, and display the unselected items.

Step 6: Click *OK* to close the dialog.

If Filter Unselected is selected, the analyzing panel is refreshed to show the selected items only (and hide the unselected items).

Figure 4.26: Result from Filter Unselected
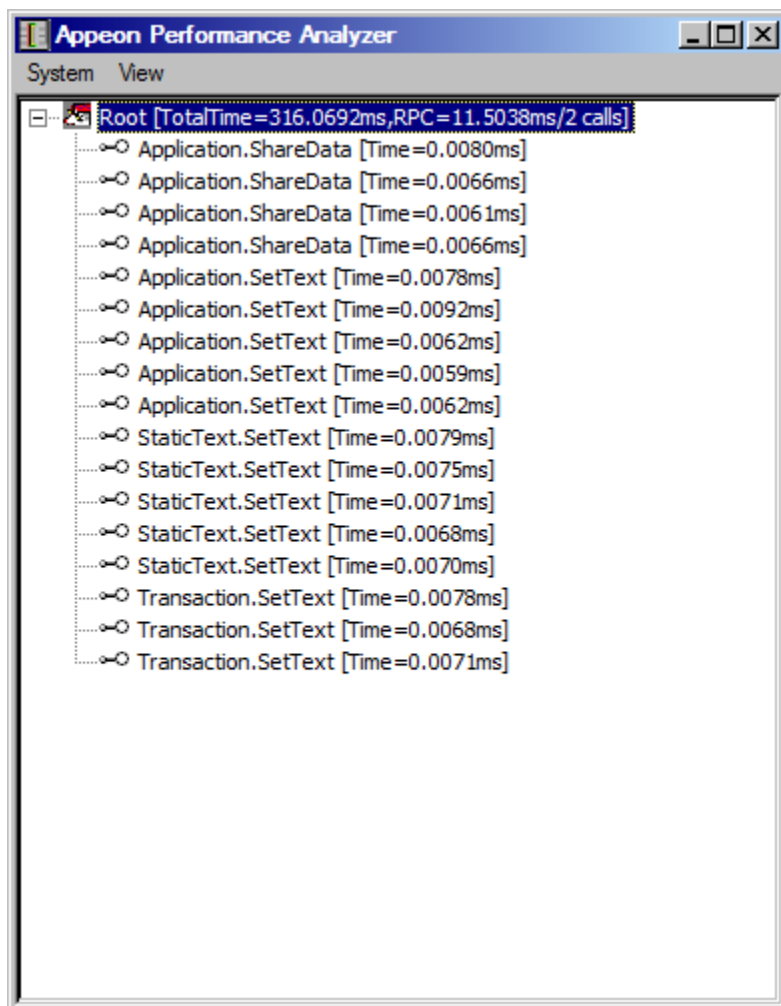
If Filter Selected is selected, the analyzing panel is refreshed to show the unselected items only (and hind the selected items).

Figure 4.27: Result from Filter selected
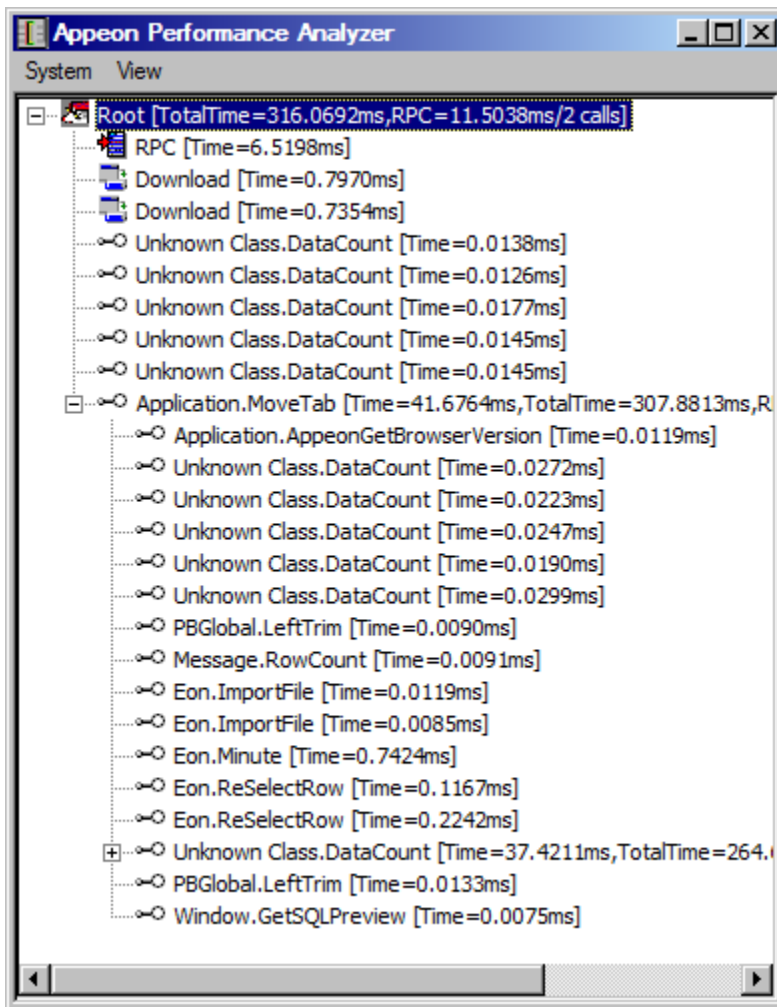
4.2.2.5.2 Find function

You can find functions or objects by typing part of or complete name of the function or object.

If you click menu System | Find, a dialog box will pop up as below:

Figure 4.28: Find dialog
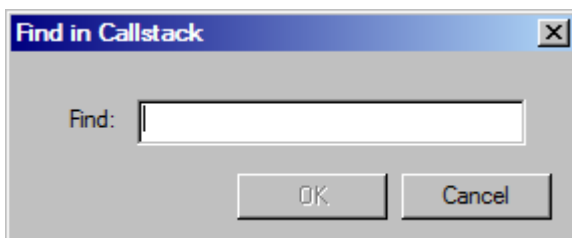
For example, type "applica" in the Find field, then click *Find*.

Figure 4.29: Find dialog
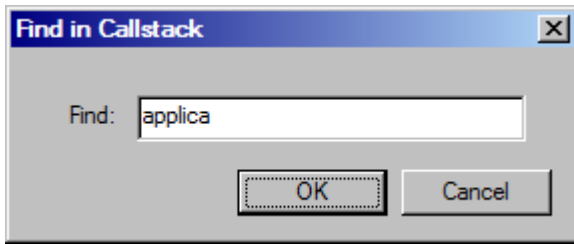
The Detail View window shows all the results related to "applica" in the Find in Tree tab.

Figure 4.30: Find results
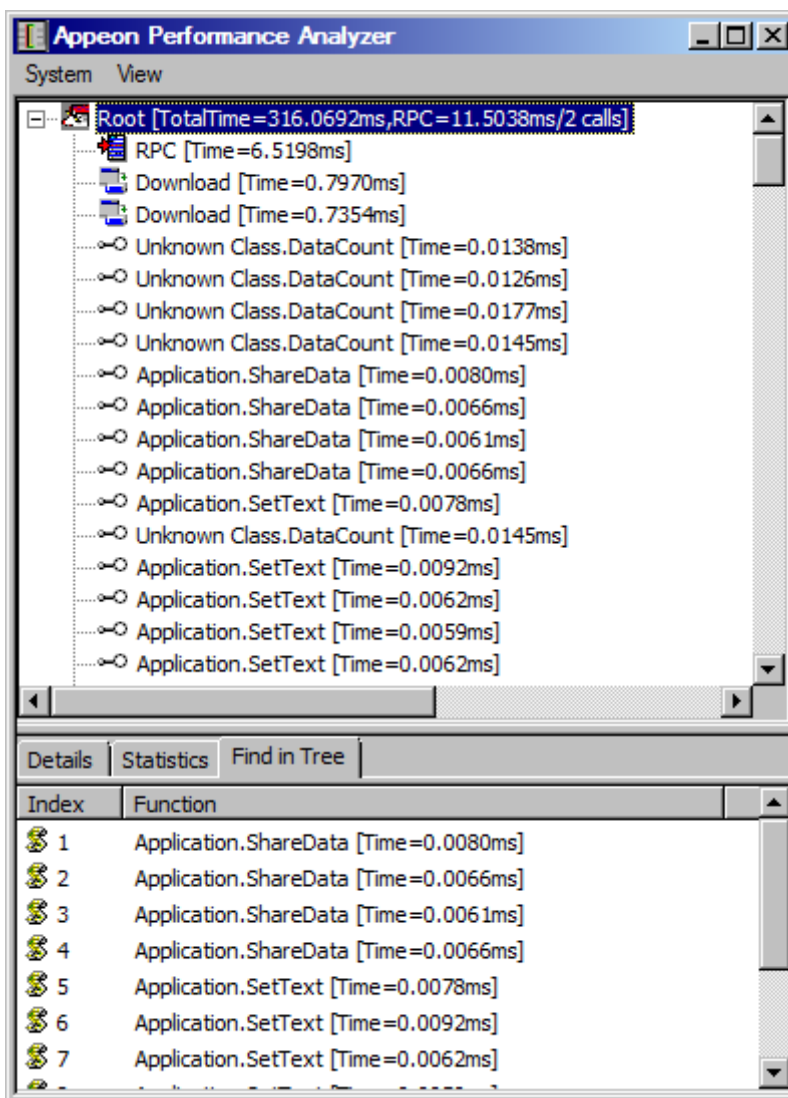
4.2.2.5.3 Save As XML function

The Performance Analyzer allows you to save the analysis contents in the Analyzing panel as XML files.

If you click menu System | Save As XML, a dialog box will pop up as below:

Figure 4.31: Save As dialog
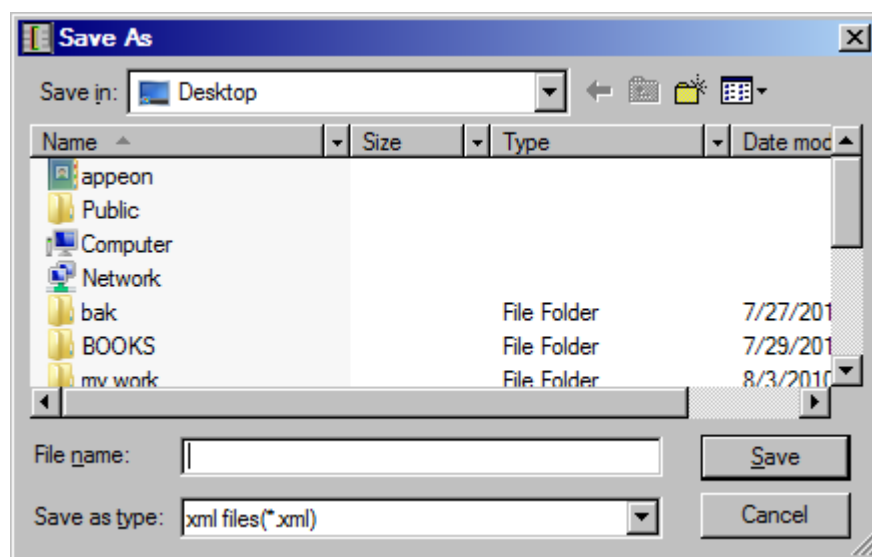
Input the file name and select the path, and then click the *Save* button to save the XML file.

4.2.2.5.4 New Root Item function

To obverse analysis data easily, you can click menu View | Clear All to clear the original analysis data, or create new root items.

If you click menu System | New Root Item, a dialog box will pop up as below:

Figure 4.32: New Root Item dialog
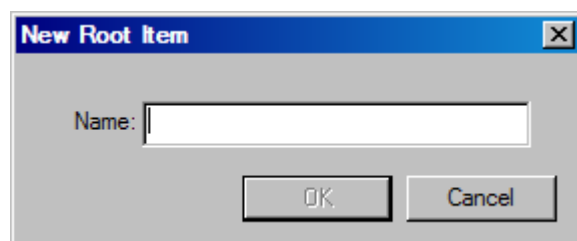
Input a root item name (eg. new) and click the *OK* button. A new root item named "new" is added in the Analysis panel.

Figure 4.33: New root item is added
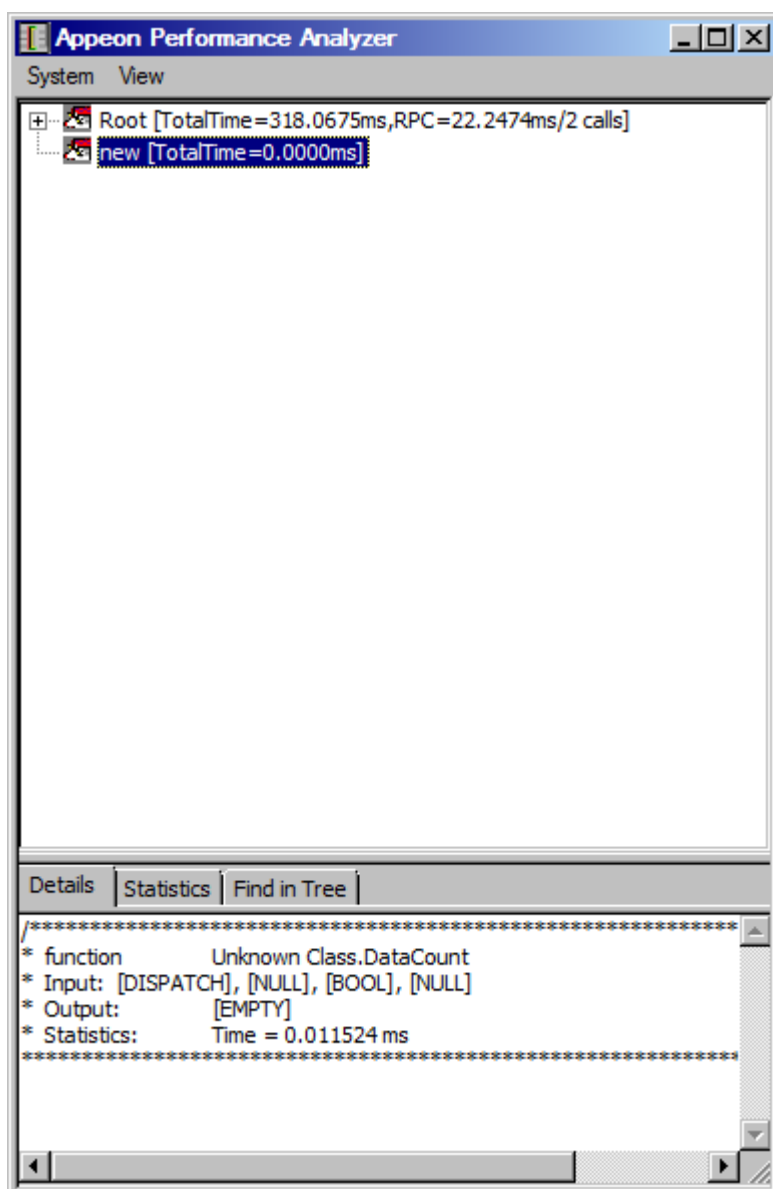
After this, the root item "new" will be used to display the analysis data in a way similar to the default Root item.

Figure 4.34: Data in the new root item
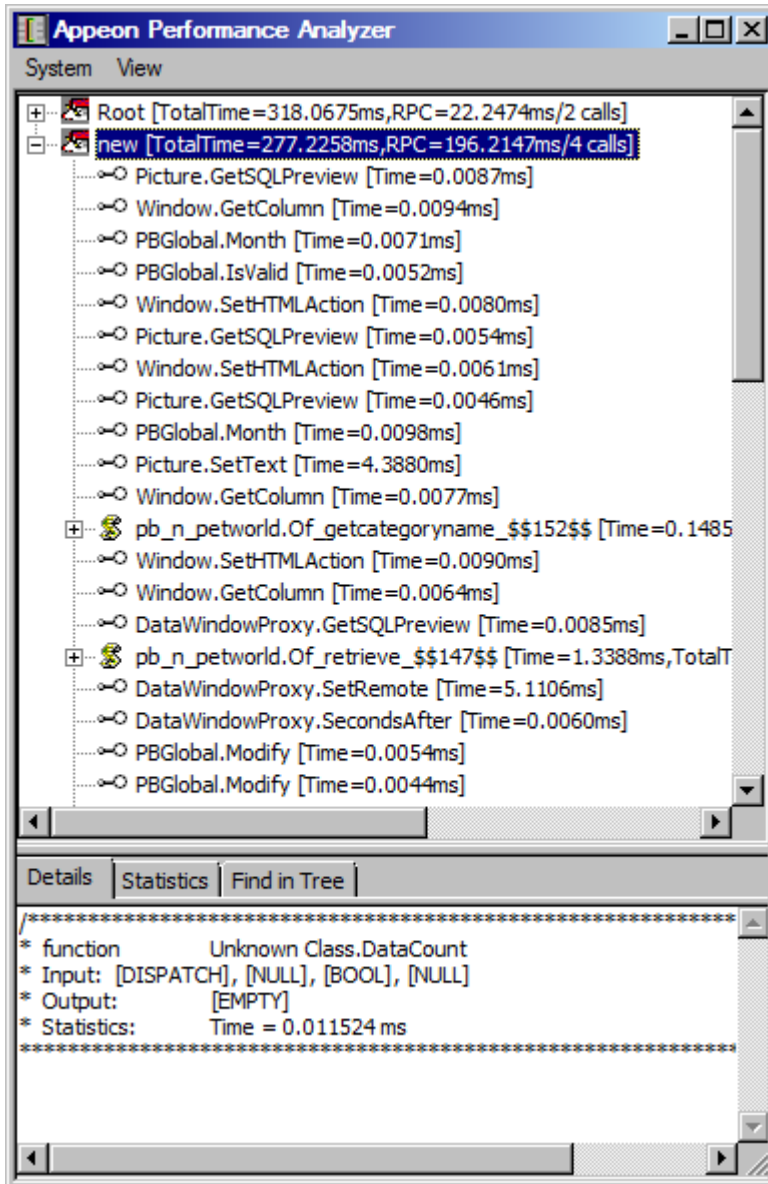
4.3 Testing Appeon Web applications with LoadRunner

4.3.1 LoadRunner

The HP LoadRunner is an automated performance and testing product for examining system behaviors and performance. The HP LoadRunner can simulate hundreds or thousands of concurrent users to put the application through the rigors of real-life user loads, while collecting information from key infrastructure components (Web servers, database servers, etc.). The results can then be analyzed in details, to explore the reasons for a particular behavior.

4.3.2 General Limitations on Performance Testing

1. The Web/HTTP protocol is used;

2. Only Web applications in English can be tested with LoadRunner.

4.3.3 Testing Steps

To put it concisely, performing test of Appeon for PowerBuilder with the HTTP protocol in the test mode is to complete the following steps:

1. Set the record parameters and record the scripts;

2. Change the constants of Session ID in the scripts to variables;

3. If testing the Update operation, you need to parameterize SQL statements (to avoid deadlock of database tables when there are many concurrent access);

4. Configure the Run-Time Settings;

5. Play back the scripts unless the tests succeed, otherwise return to 2), 3), 4);

6. Design and run the scenario according to the testing scheme.

4.3.3.1 Configuring AEM

1. Select the **Test Mode** as the run mode:

   Navigate to **AEM** > **Application** > **Client Features** > YourApplicationName > **Run Mode** > **Test Mode**, and then click **Save**.

2. Select the **Debug Mode** as the log generation mode:

   Navigate to **AEM** > **Server** > **Logging** > **Log Mode** > **Debug Mode**, and then click **Save**.

After you play back scripts successfully in vugen.exe, change the **Debug Mode** back to **Standard Mode**.

4.3.3.2 Data Preparation (for update only)

This is required only when testing the Update operation.

To simulate large number of concurrent access of Update, sufficient records should be prepared first.

Requirements in the modified table:

1. The Recording Numbers must meet the following condition:

   In the table, the number of Records should be greater than the number of (Vusers + 1) times the number of Modified Record/Users (Number of Records > (Vusers + 1) * Number of Modified Record/Users).

   For example, if there are 10 users concurrently accessing and conducting the DataWindow Update 10r x 7c with the "use update" mode operation, the modified table must at least contain the following number of records: (10 + 1) * 10 = 110.

2. The initial value of the keyword "id" should be 0 (This is for simplifying the code process for the SQL statement parameterization in scripts).

4.3.3.3 Preparing Test Cases

Determine the scripts to be recorded and the actions to be executed in scripts.

4.3.3.4 Recording Scripts

1. Set the record engine: **Record script using earlier recording engine**.

   Select from **File** > **Tools** > **Recording Options** > **Advanced**.

   Check the **Record script using earlier recording engine** checkbox below **Recording Engine**, and then click **OK** as shown in the following figure.

   Figure 4.35: Recording Options
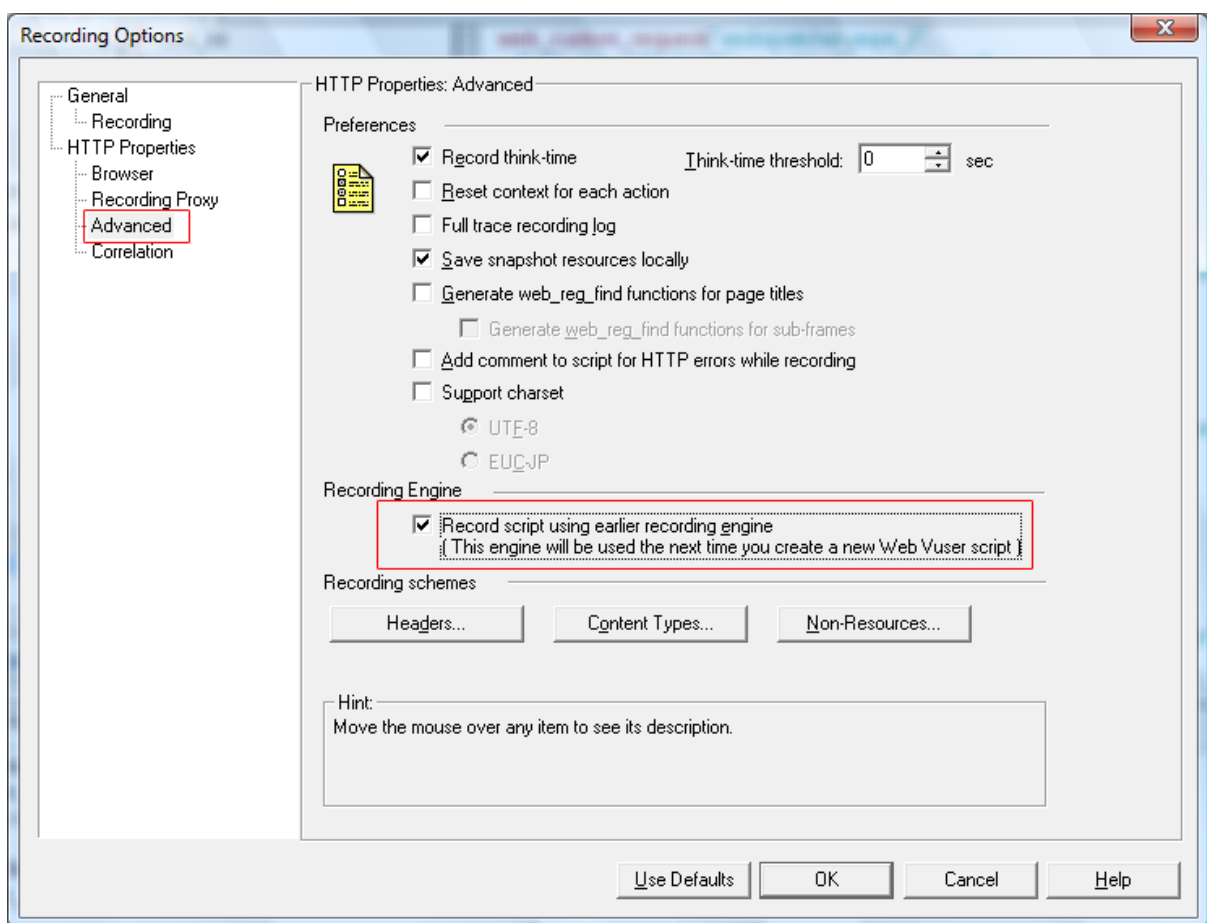
   Restart the **LoadRunner**.

2. Set the protocol for recording: **Web (HTTP/HTML)**.

   Select from **File**> **New**.

   In the **New Virtual User** window, select **Web (HTTP/HTML)**, and then click **OK** as shown in the following figure.

Figure 4.36: New Virtual User
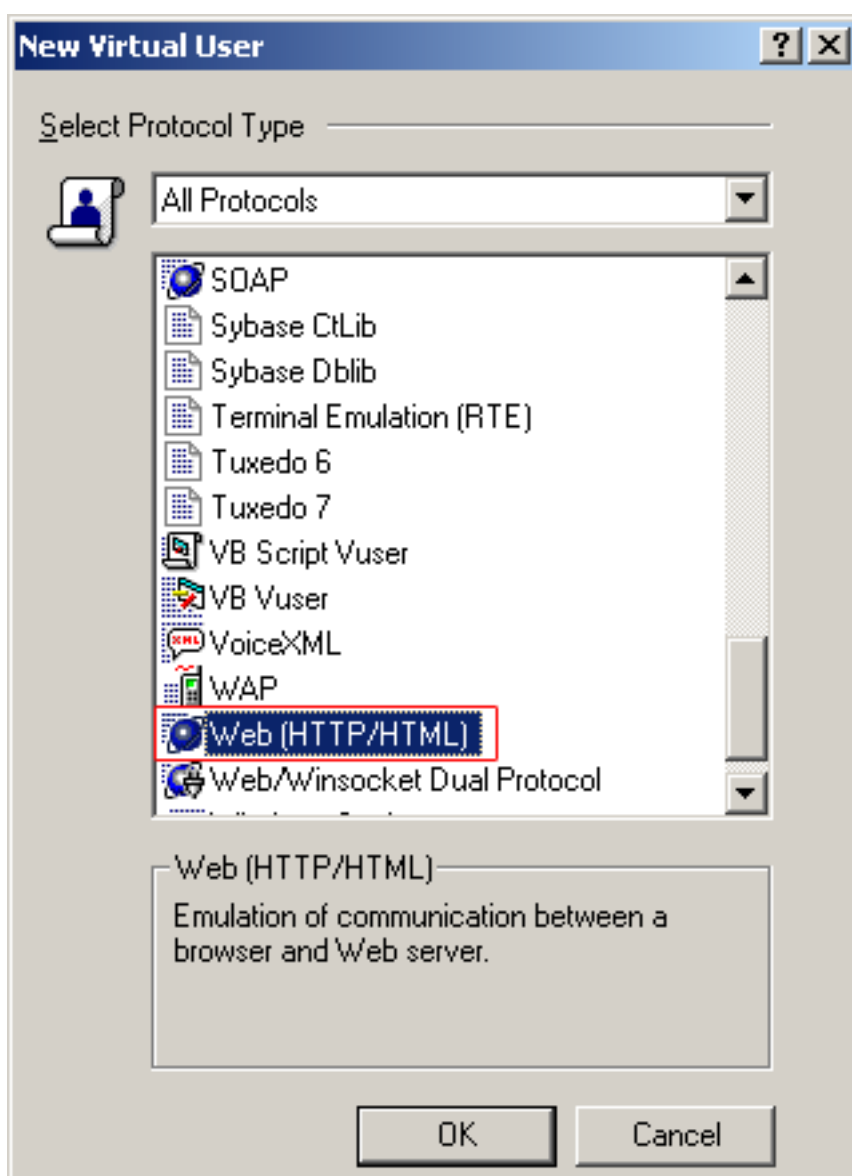
3. Set the **Recording** options.

    Select from **Tools** > **Recording Options**.

    In the window that appears, select **General** > **Recording** on the left, and then select the **URL-based scripts** radio button as shown in the following figure.

Figure 4.37: Recording Options
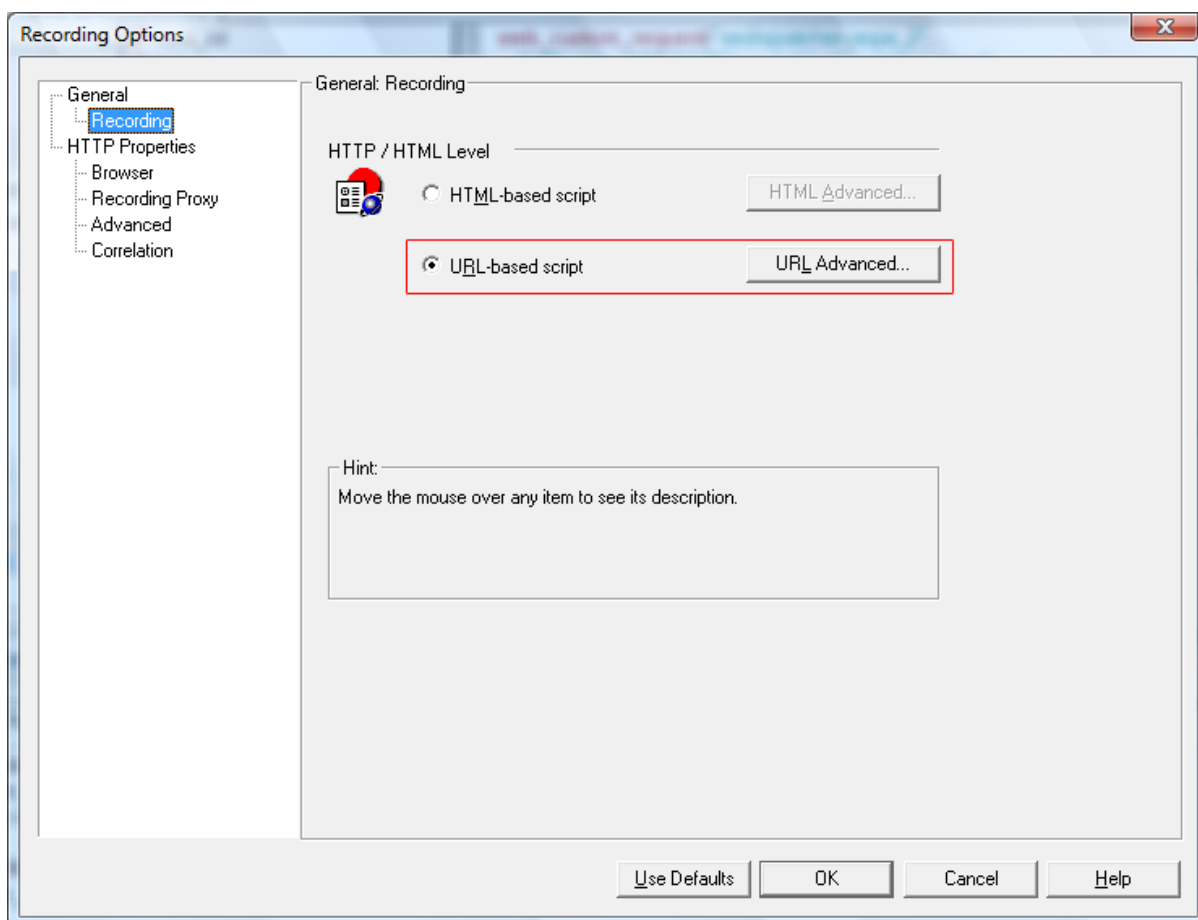
Click the **URL Advanced** button in the same window, and the **Advanced URL** window appears.

Check all the checkboxes and click **OK** as shown in the following figure.

Figure 4.38: Advanced URL
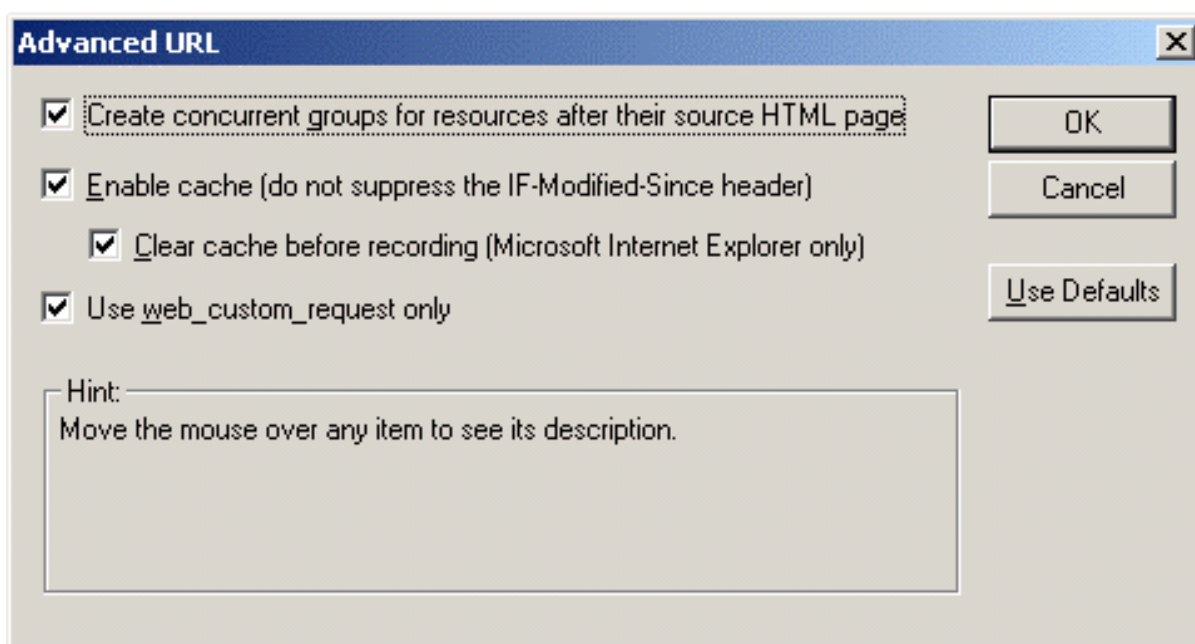
4. Set the **Advanced** options.

   Select from **Tools** > **Recording Options**.

   In the window that appears, select **HTTP Properties** > **Advanced** on the left, and then uncheck the **Reset context for each action** option, as shown in the following figure.

Figure 4.39: Recording Options
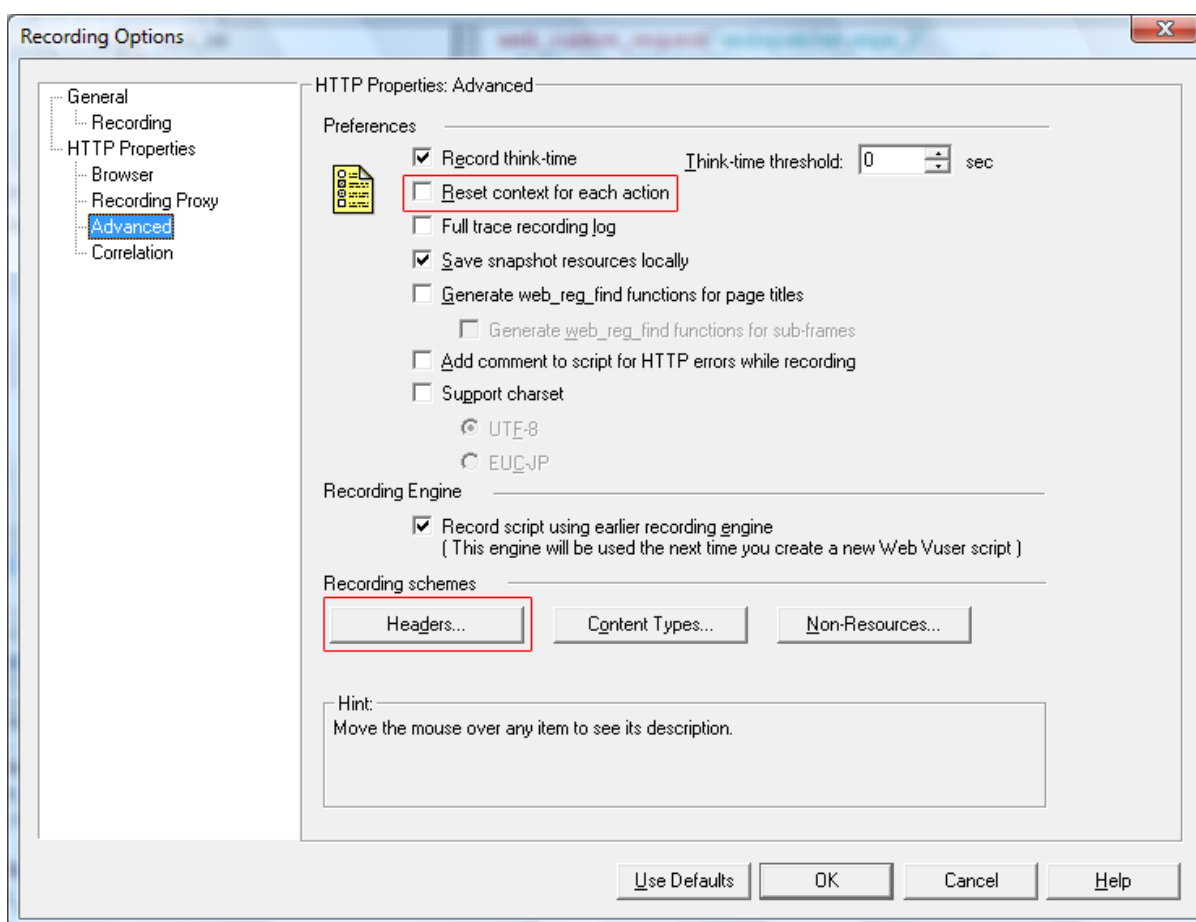
Click the **Headers** button in the same window.

In the **Headers** window, select the **Record headers in list** from the dropdown list box, and then check the **Accept** and the **Accept-Language** checkboxes, as shown in the following figure.

Figure 4.40: Headers
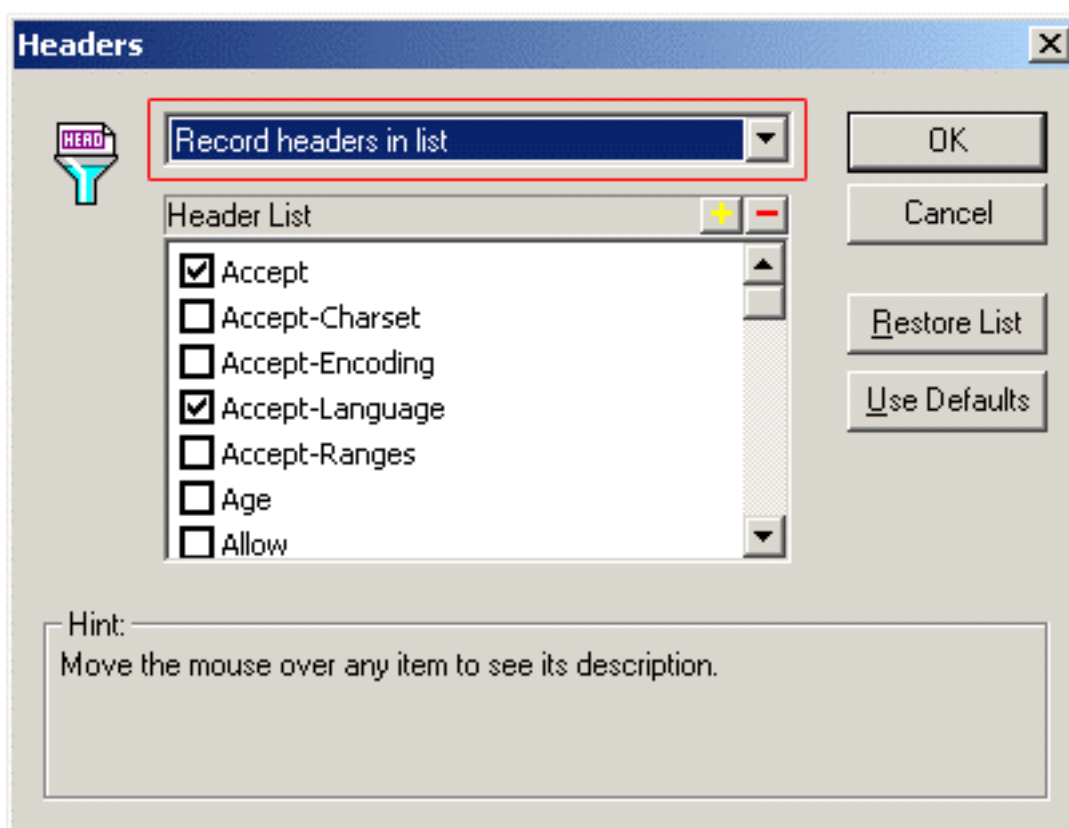
Click the "+" sign to add two headers: **pt** and **ProtocolMode**, then select them in the **Header List**, and then click **OK** as shown in the following figure.

Figure 4.41: Headers
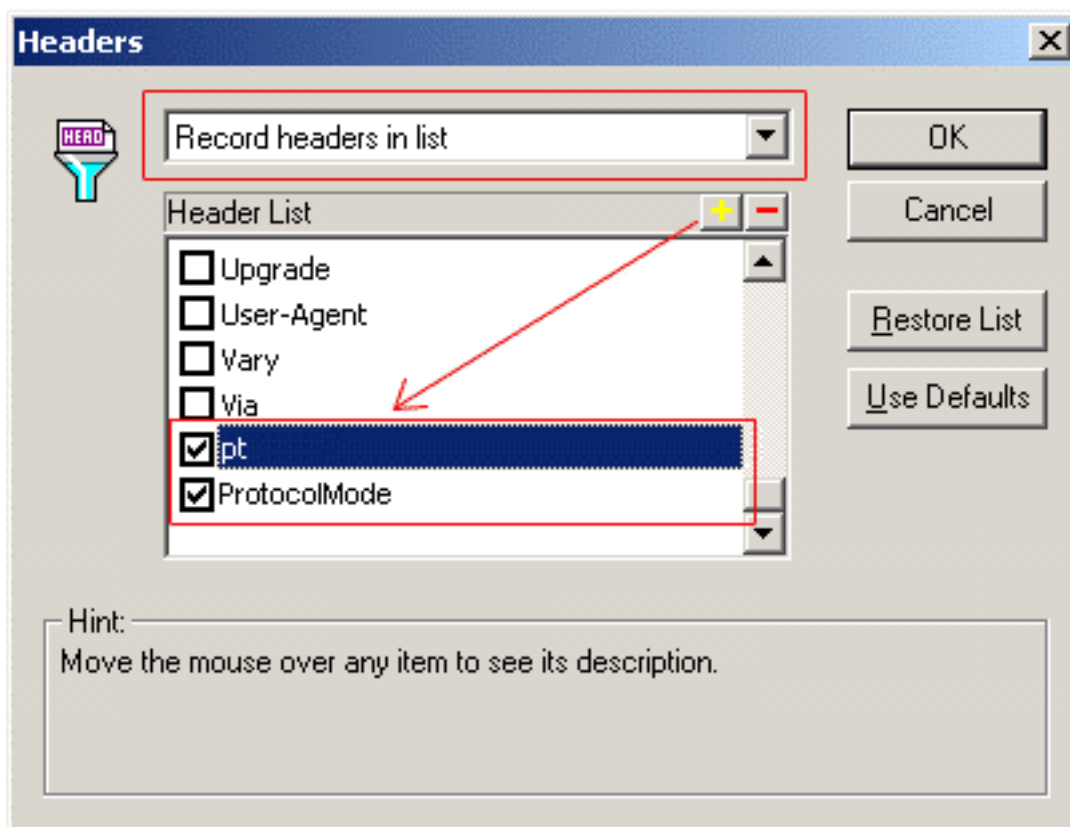
4.3.3.5 Modifying Scripts

Parameterize scripts of SessionID:

1. Use the keyword "servlet" to search in the scripts, and add the following scripts to the line previous to the line where the second "servlet" appears, so as to capture the current SessionID.

```
web_reg_save_param("SessionID","LB=ID =","RB=\r\n",LAST);
```

2. Search all the remaining "servlet" statements, and modify the parameters starting with "Body=" in the following way:

Find the location of "Session-ID =", and replace the recorded value of SessionID with the character string "{SessionID}", for example,

Replace

```
"Session-ID = -318005152\r\n"
```

with

```
"Session-ID = {SessionID}\r\n"
```

The following block (*{SessionID} in "Session-ID = {SessionID}\r\n"*) is an example after changed.

```
web_custom_request("appeon.weblibrary.AXDispatcher_6",
"URL=http://172.17.1.108:8080/servlet/appeon.weblibrary.AXDispatcher",
"Method=POST",
"Resource=1",
"Referer=",
"EncType=",
"Body="
"\\x1F\\x8B\\x08\\x00\\x00\\x00\\x00\\x00\\x00\\x0B"
"ST"
"\\x16"
"b0a0"
"\\x06"
"b"
"\\x0B\\x06"
"K"
"\\x06"
"C"
"\\x06"
"3"
"\\x06"
"#"
"\\x06"
"S"
"\\x06"
"1"
"\\x86\\x00"
"†T"
"\\x86"
"\""
"\\x86"
"4†| ™Ë#È#Ç"
"\\x90\\x0C\\x14\\x01\\x01"
"1 fb"
"\\x00\\x00\\x12\\x02"
"½#8"
"\\x00\\x00\\x00"
"-------------------appeon data\r\n"
"Name = Kill Session\r\n"
"Session-ID = {SessionID}\r\n"
"-------------------appeon data\r\n"
"D"
"\\x00\\x00\\x00"
,
LAST);
```

4.3.3.5.1 Additional steps for the Update operation

For the Update operation, the script parameterization should meet the following three requirements:

1. Parameterization of SessionID (which has already been described in Modifying Scripts);

2. Parameterization of SQL statements (which will be described in Parameterization of DataWindow SQL statements and in Parameterization of Embedded SQL statements);

3. Add the Commit request to the end of each Update request, that is, move the Commit request in the recorded scripts from end.c to action.c.

4.3.3.5.2 Parameterization of DataWindow SQL statements

1. The recorded DataWindow SQL statements are as follows:

```
"UpDateSQL_1 = update \"PERF_TEST1\" set \"depid\" =
1,\"code\" = 'modify1',\"name\" = 'modify1',\"born\" = '2006-09-04
14:35:12.843',\"age\" = 1,\"salary\" = 1 where \"id\" = 0;update
\"PERF_TEST1\" set \"depid\" = 1,\"code\" = 'modify1',\"name\" =
'modify1',\"born\" = '2006-09-04 14:35:12.843',\"age\" =
1,\"salary\" = 1 where \"id\" = 1;update \"PERF_TEST1\" set
\"depid\" = 1,\"code\" = 'modify1',\"name\" = 'modify1',\"born\" =
'2006-09-04 14:35:12.843',\"age\" = 1,\"salary\" = 1 where \"id\"
= 2;update \"PERF_TEST1\" set \"depid\" = 1,\"code\" =
'modify1',\"name\" = 'modify1',\"born\" = '2006-09-04
14:35:12.843',\"age\" = 1,\"salary\" = 1 where \"id\" = 3;update
\"PERF_TEST1\" set \"depid\" = 1,\"code\" = 'modify1',\"name\"
= 'modify1',\"born\" = '2006-09-04 14:35:12.843',\"age\" =
1,\"salary\" = 1 where \"id\" = 4;update \"PERF_TEST1\" set
\"depid\" = 1,\"code\" = 'modify1',\"name\" = 'modify1',\"born\" =
'2006-09-04 14:35:12.859',\"age\" = 1,\"salary\" = 1 where \"id\"
= 5;update \"PERF_TEST1\" set \"depi"
"d\" = 1,\"code\" = 'modify1',\"name\" =
'modify1',\"born\" = '2006-09-04 14:35:12.859',\"age\" =
1,\"salary\" = 1 where \"id\" = 6;update \"PERF_TEST1\" set
\"depid\" = 1,\"code\" = 'modify1',\"name\" = 'modify1',\"born\" =
'2006-09-04 14:35:12.859',\"age\" = 1,\"salary\" = 1 where \"id\"
= 7;update \"PERF_TEST1\" set \"depid\" = 1,\"code\" =
'modify1',\"name\" = 'modify1',\"born\" = '2006-09-04
14:35:12.859',\"age\" = 1,\"salary\" = 1 where \"id\" = 8;update
\"PERF_TEST1\" set \"depid\" = 1,\"code\" = 'modify1',\"name\" =
'modify1',\"born\" = '2006-09-04 14:35:12.859',\"age\" =
1,\"salary\" = 1 where \"id\" = 9\r\n"
```

a. To make the field value change every time the same user executes it, you need to parameterize the constant after the field: "modify1" in the above example.

b. To ensure that different virtual users modify different records, you need to parameterize the id value after the "Where" keyword.

For these purposes, at least two parameters should be set: "Modified" and "ID".

To parameterize fields of the number type and the data type, it is also necessary to set the new parameters.

In this example, the type of parameter "Modified" is iteration number, and the type of parameter "id" is UserID.

2. In the scripts, for most field values, you must perform the following two replacements in the DataWindow SQL statements.

a. Replace "modify1" with "{modified}"

b. Replace "where \"id\" = " (**Note**: In this example, the last character included in the quotation mark must be a space) with "where \"id\" = {id}"

If the text to be replaced is split and displayed in two lines, for example, "modify1" is displayed as "modif" and "y1" in two separated lines, you need to manually modify or programmatically replace it, otherwise transaction failures will occur when there are many concurrent user access.

Make sure that every SQL statement is correctly replaced.

```
································,\"col8\" = '{modified}',\"col9\" = 'modif"
"y1',\"col10\" = 'modify1' where \"id\" = {id}06;·····················
```

3. The parameterized SQL statements are as follows:

```
"UpDateSQL_1 = update \"PERF_TEST1\" set \"depid\" = 1,\"code\" =
'{modified}',\"name\" = '{modified}',\"born\" = '2006-09-04
11:25:12.859',\"age\" = 1,\"salary\" = 1 where \"id\" =
{id}0;update \"PERF_TEST1\" set \"depid\" = 1,\"code\" =
'{modified}',\"name\" = '{modified}',\"born\" = '2006-09-04
11:25:12.859',\"age\" = 1,\"salary\" = 1 where \"id\" =
{id}1;update \"PERF_TEST1\" set \"depid\" = 1,\"code\" =
'{modified}',\"name\" = '{modified}',\"born\" = '2006-09-04
11:25:12.859',\"age\" = 1,\"salary\" = 1 where \"id\" =
{id}2;update \"PERF_TEST1\" set \"depid\" = 1,\"code\" =
'{modified}',\"name\" = '{modified}',\"born\" = '2006-09-04
11:25:12.875',\"age\" = 1,\"salary\" = 1 where \"id\" =
{id}3;update \"PERF_TEST1\" set \"depid\" = 1,\"code\" =
'{modified}',\"name\" = '{modified}',\"born\" = '2006-09-04
11:25:12.875',\"age\" = 1,\"salary\" = 1 where \"id\" =
{id}4;update \"PERF_TEST1\" set \"depid\" = 1,\"code\" =
'{modified}',\"name\" = '{modified}',\"born\" = '2006-09-04
11:25:12.875',\"age\" = 1,\"salary\" = 1 where \"id\" =
{id}5;update \"PERF_TEST1\" set \"depid\" = 1,\"code\" =
'{modified}',\"name\" = '{modified}',\"born\" = '2006-09-04
11:25:12.875',\"age\" = 1,\"salary\" = 1 where \"id\" =
{id}6;update \"PERF_TEST1\" set \"depid\" = 1,\"code\" =
'{modified}',\"name\" = '{modified}',\"born\" = '2006-09-04
11:25:12.875',\"age\" = 1,\"salary\" = 1 where \"id\" =
{id}7;update \"PERF_TEST1\" set \"depid\" = 1,\"code\" =
'{modified}',\"name\" = '{modified}',\"born\" = '2006-09-04
11:25:12.890',\"age\" = 1,\"salary\" = 1 where \"id\" =
{id}8;update \"PERF_TEST1\" set \"depid\" = 1,\"code\" =
'{modified}',\"name\" = '{modified}',\"born\" = '2006-09-04
11:25:12.890',\"age\" = 1,\"salary\" = 1 where \"id\" = {id}9\r\n"
```

4. When playing back the scripts after replacement, the DataWindow SQL statements
   executed are as follows:

```
UpDateSQL_1 = update "PERF_TEST1" set "depid" = 1,"code" =
'modify1',"name" = 'modify1',"born" = '2006-09-04
11:25:12.859',"age" = 1,"salary" = 1 where "id" =
10;update "PERF_TEST1" set "depid" = 1,"code" =
'modify1',"name" = 'modify1',"born" = '2006-09-04
11:25:12.859',"age" = 1,"salary" = 1 where "id" =
11;update "PERF_TEST1" set "depid" = 1,"code" =
'modify1',"name" = 'modify1',"born" = '2006-09-04
11:25:12.859',"age" = 1,"salary" = 1 where "id" =
12;update "PERF_TEST1" set "depid" = 1,"code" =
'modify1',"name" = 'modify1',"born" = '2006-09-04
11:25:12.875',"age" = 1,"salary" = 1 where "id" =
13;update "PERF_TEST1" set "depid" = 1,"code" =
'modify1',"name" = 'modify1',"born" = '2006-09-04
11:25:12.875',"age" = 1,"salary" = 1 where "id" =
14;update "PERF_TEST1" set "depid" = 1,"code" =
'modify1',"name" = 'modify1',"born" = '2006-09-04
11:25:12.875',"age" = 1,"salary" = 1 where "id" =
15;update "PERF_TEST1" set "depid" = 1,"code" =
'modify1',"name" = 'modify1',"born" = '2006-09-04
11:25:12.875',"age" = 1,"salary" = 1 where "id" =
16;update "PERF_TEST1" set "depid" = 1,"code" =
'modify1',"name" = 'modify1',"born" = '2006-09-04
11:25:12.875',"age" = 1,"salary" = 1 where "id" =
17;update "PERF_TEST1" set "depid" = 1,"code" =
'modify1',"name" = 'modify1',"born" = '2006-09-04
11:25:12.890',"age" = 1,"salary" = 1 where "id" =
18;update "PERF_TEST1" set "depid" = 1,"code" =
'modify1',"name" = 'modify1',"born" = '2006-09-04
11:25:12.890',"age" = 1,"salary" = 1 where "id" = 19
```

5. In the iterative operations, the DataWindow SQL statements executed at the third time are as follows:

```
UpDateSQL_1 = update "PERF_TEST1" set "depid" = 1,"code" =
'modify3',"name" = 'modify3',"born" = '2006-09-04
11:25:12.859',"age" = 1,"salary" = 1 where "id" =
10;update "PERF_TEST1" set "depid" = 1,"code" =
'modify3',"name" = 'modify3',"born" = '2006-09-04
11:25:12.859',"age" = 1,"salary" = 1 where "id" =
11;update "PERF_TEST1" set "depid" = 1,"code" =
'modify3',"name" = 'modify3',"born" = '2006-09-04
11:25:12.859',"age" = 1,"salary" = 1 where "id" =
12;update "PERF_TEST1" set "depid" = 1,"code" =
'modify3',"name" = 'modify3',"born" = '2006-09-04
11:25:12.875',"age" = 1,"salary" = 1 where "id" =
13;update "PERF_TEST1" set "depid" = 1,"code" =
'modify3',"name" = 'modify3',"born" = '2006-09-04
11:25:12.875',"age" = 1,"salary" = 1 where "id" =
14;update "PERF_TEST1" set "depid" = 1,"code" =
'modify3',"name" = 'modify3',"born" = '2006-09-04
11:25:12.875',"age" = 1,"salary" = 1 where "id" =
15;update "PERF_TEST1" set "depid" = 1,"code" =
'modify3',"name" = 'modify3',"born" = '2006-09-04
11:25:12.875',"age" = 1,"salary" = 1 where "id" =
16;update "PERF_TEST1" set "depid" = 1,"code" =
'modify3',"name" = 'modify3',"born" = '2006-09-04
11:25:12.875',"age" = 1,"salary" = 1 where "id" =
17;update "PERF_TEST1" set "depid" = 1,"code" =
'modify3',"name" = 'modify3',"born" = '2006-09-04
11:25:12.890',"age" = 1,"salary" = 1 where "id" =
18;update "PERF_TEST1" set "depid" = 1,"code" =
'modify3',"name" = 'modify3',"born" = '2006-09-04
11:25:12.890',"age" = 1,"salary" = 1 where "id" = 19
```

In this way, the value changes every time the same user executes an Update operation.

4.3.3.5.3 Parameterization of Embedded SQL statements

To parameterize the value in the embedded SQL statement, you will have to define the value as a variable.

For example, the following script cannot be parameterized.

```
select a from table where c = '1';
```

You will have to modify the script as below, in order to parameterize the value:

```
int ic
ic = 1
select a from table where c= :ic
```

4.3.3.6 Playing back scripts to test the correctness of scripts

Before playing back the scripts, set as follows from the menu: **Vuser** > **Run-time Settings** > **Browser Emulation** > **Options** > **Check new resources when simulating browser cache** = **Yes**.

Check the log of AEM to see if the log of recorded scripts is the same as the log of playback (SessionID values can be different), otherwise, go back to the previous step to check and modify the scripts and settings again.

4.3.3.7 Setting Scenarios

1. Select from **Tools** > **Create controller scenario**;

2. Set the schedule of scenario according to your needs;

3. Configure the parameter settings for scenario:

   Navigate to **Browser Emulation**, uncheck the **Simulate browser cache** and the **Simulate a new user each iteration** checkboxes, and then check the **Download non-HTML resources** checkbox, as shown in the following figure.

   Click **OK**.

Figure 4.42: Run-times Settings
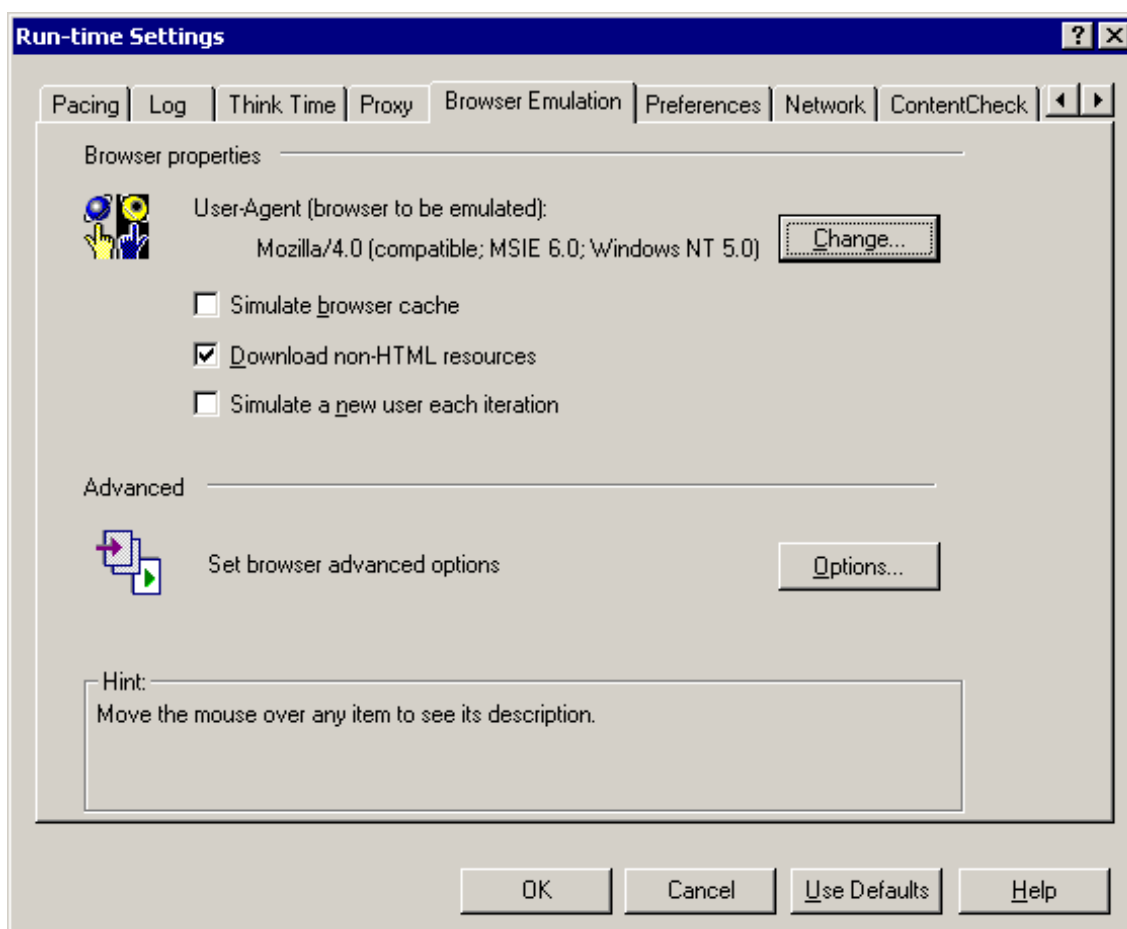
4.3.3.7.1 Additional steps for the Update operation

Set the Update operation being committed when 10 virtual users access concurrently. Make sure that the IDs of virtual users are the successive integers starting from 1, for example, 1~10, and keep running for 5 minutes.

When you run the scenario, ensure that it runs correctly without any error or error messages in the AEM log, and check the corresponding tables in the database and make sure the values of records (id 10-109) have been changed.

4.3.3.8 Running Scenarios

Select to run the scenario and the test report will be generated.

4.3.4 Troubleshooting

4.3.4.1 "The session does not exist" error

Phenomena: The scripts run correctly right after they are recorded and modified, but after a while or after the server is restarted, the following error messages appear in the log of Appeon when playing back the scripts:

```
1150101302078:0 [06-06-12 16:35:02.078]
[com.appeon.server.services.controller.JRouterImpl (call())] The
session does not exist."
```

Solution: Check and make sure that the **Reset context for each action** option is unchecked as shown in the following figure, and then record the scripts again.

In this way, the scripts will remain usable even after the application server is restarted.

Figure 4.43: Recording Options
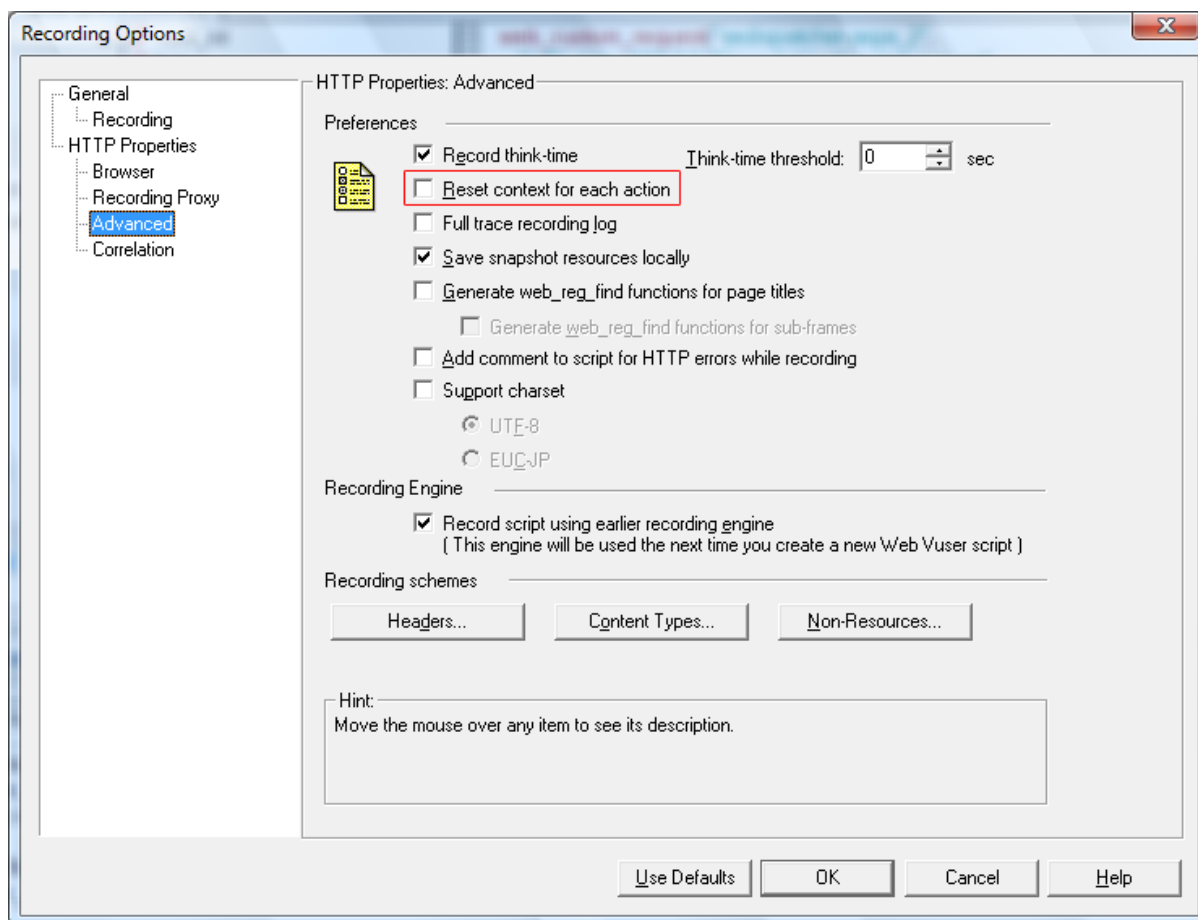
4.3.4.2 Errors appear when playing back scripts with LoadRunner 8.0

Phenomena: The following error messages will appear in the ApeonServer.log or AppeonError.log when the scripts are played back with LoadRunner 8.0:

```
1154707726562:0 [06-08-04 09:08:46.562]
[com.appeon.server.services.controller.JRouterImpl (call())] The
 session does not exist.
```

Solution: This is a known issue with LoadRunner 8.0.

Please upgrade LoadRunner 8.0 to a later version.

4.3.4.3 The value of sessionID is null

Phenomena: If you use the following statements to debug the value of SessionID, the SessionID value will be null in the Execution Log after the script is played back.

```
lr_output_message(lr_eval_string("SessionID-------->"));
lr_output_message(lr_eval_string("{SessionID}"));
```

Solution: The function (as follows)

```
web_reg_save_param("SessionID","LB=ID =","RB=\r\n",LAST);
```

to obtain Session ID should be placed before the second HTTP request. Do not place it before the first request.

```
web_custom_request("AXDispatcher.aspx_2", "URL=http://172.17.1.29/servlet/ AXDispa
  tcher",
  "Method=POST",
  "Resource=1",
  "RecContentType=text/plain",
  "Referer=",
  "Binary=1",
  "EncType=",
  "Body="
  "\\x1F\\x8B\\x08\\x00\\x00\\x00\\x00\\x00\\x00\\x0B"
  "SPf`"
  "\\xD0"
  "cHd("
  "\\x00\\xC2"
  "T"
  "\\x86"
  "|"
  "\\x86"
  "<"
  "\\x86"
  "x"
  "\\x86"
  "d "
  "\\x9D\\x02\\xE4\\xC5\\x03"
  "q"
  "\\x05"
  "P."
  "\\x17"
  "("
  "\\x97\\x03"
  "d"
  "\\x17\\x03"
  "E"
  "\\x12\\x81"
  "\""
  "\\x8C\\x0C"
  "@"
  "\\xC0\\x02\\xC4"
  "L"
  "\\x0C\\x00\\x9D\\x84\\xA2\\xD6"
  "?"
  "\\x00\\x00\\x00"
  "",
  LAST);
```

4.3.4.4 Error message appears in script playback

Phenomena: After recording, modifying, and playing back the scripts, the following messages appear in the Execution log.

```
vuser_init.c(236): Error: No match found for the requested
parameter "SessionID". If the data you want to save exceeds 256
bytes, use web_set_max_html_param_len to increase the parameter size
vuser_init.c(236): Error: The above "not found" error(s) may be
explained by header and body byte counts being 0 and 0, respectively.
Abort was called from an action
```

Solution: This is probably caused by the Web cache setting. If you will not test Web cache, be sure to cancel the settings for simulating cache in the **Run-time Settings**, by selecting from **Vuser** > **Run-time Settings** > **Browser Emulation** > **Options** > **Check for newer resources when simulating browser cache** = **Yes**.

4.3.4.5 Error message in Appeon Log

Phenomena: After the script is played back, set the pacing>1 and play back the scripts again, the following error messages appear in the AppeonServer log for AEM:

```
1154707726562:0 [06-08-04 09:08:46.562]
[com.appeon.server.services.controller.JRouterImpl (call())]
The session does not exist.
1154707726593:0 [06-08-04 09:08:46.593]
[com.appeon.server.services.controller.JRouterImpl (call())]
The session does not exist.
1154707726625:0 [06-08-04 09:08:46.625]
[com.appeon.server.services.controller.JRouterImpl (call())]
The session does not exist.
1154707726687:0 [06-08-04 09:08:46.687]
[com.appeon.server.services.controller.JRouterImpl (call())]
The session does not exist.
1154707726718:0 [06-08-04 09:08:46.718]
[com.appeon.server.services.controller.JRouterImpl (call())]
The session does not exist.
```

Solution: Make sure that the **Simulate a new user each iteration** option for **Run-time Settings** is unchecked.

4.3.4.6 Failed to parameterize scripts

Phenomena: After the scripts are recorded, the plain text cannot be found in all web_custom_request as shown below:

```
web_custom_request("appeon.weblibrary.AXDispatcher3",
  "URL=http://172.17.1.29/servlet/appeon.weblibrary.AXDispatcher",
  "Method=POST",
  "Resource=1",
  "RecContentType=text/plain",
  "Referer=",
  "Binary=1",
  "EncType=",
  "Body="
  "\\x1F\\x8B\\x08\\x00\\x00\\x00\\x00\\x00\\x00\\x0B"
  "kQ"
  "\\x16"
  "a"
  "\\xD0"
  "e"
  "\\xB0"
  "d0c0e0"
  "\\x07\\xD3\\xE6\\x0C"
  "&"
  "\\x0C\\xC6\\x0C\\x5C\\xC5\\x0C\\x85\\x0C"
  "9"
  "\\x0C\\xC9\\x0C\\x89\\x0C\\x5C\\x0C\\xFF\\x19\\x19\\x98\\x18"
  "Y"
  "\\x18"
  "8"
  "\\x81"
  "Ln"
  "\\x06\\x1E\\x06"
  "^"
  "\\x06"
  ">"
  "\\x06\\x10\\x90\\x00"
  "b&"
  "\\x06\\x00\\xE6\\x89"
  "_xB"
  "\\x00\\x00\\x00"
  "",
  LAST);
```

Solution: Before running the scenario, make sure that you have selected **Test Mode** in the **Run Mode** settings of AEM and that you have set the log mode to **Standard**. Then the plain text ("--------------------appeon data\r\n" "Name = Connect\r\n" "Session-ID = -1476559241\r\n" "--------------------appeon data\r\n" "P" "\\x00\\x00\\x00") will appear as shown below.

```
web_custom_request("appeon.weblibrary.AXDispatcher3",
  "URL=http://172.17.1.29/servlet/appeon.weblibrary.AXDispatcher",
  "Method=POST",
  "Resource=1",
  "RecContentType=text/plain",
  "Referer=",
  "Binary=1",
  "EncType=",
  "Body="
  "\\x1F\\x8B\\x08\\x00\\x00\\x00\\x00\\x00\\x00\\x0B"
  "kQ"
  "\\x16"
  "c"
  "\\xD0"
  "e0d0a0g0c0"
  "\\x05"
  "BK"
  "\\x06"
  "# "
  "\\xCF\\x90\\x81\\xAB\\x98\\xA1\\x90"
  "!"
  "\\x87"
  "!"
  "\\x99"
  "!"
  "\\x91\\x81\\x8B\\xE1"
  "?#"
  "\\x03\\x13"
  "#"
  "\\x0B\\x03"
  "'"
  "\\x90\\xC9\\xCD\\xC0\\xC3\\xC0\\xCB\\xC0\\xC7\\x00\\x02"
  "R@"
  "\\xCC\\xC4\\x00\\x00"
  "HO"
  "\\xC9"
  "&D"
  "\\x00\\x00\\x00"
  "-------------------appeon data\r\n"
  "Name = Connect\r\n"
  "Session-ID = -1476559241\r\n"
  "-------------------appeon data\r\n"
  "P"
  "\\x00\\x00\\x00"
  "",
  LAST);
```

4.3.4.7 Out of memory error and the application server shut down

Phenomena: On the Solaris platform, WebLogic reports the out of memory error, and then shuts down. This is because the physical memory of the server is too low.

Solution: Increase the virtual memory of the server.

1. For Windows, set the virtual memory to 2G;

2. For Solaris, set the swap space to 2G.

   The commands to set swap on Solaris are:

   ```
   Mkfile /beaswap
   swap -a  /beaswap
   ```

To add new swap space, this command must be run every time a Solaris server is restarted.

4.3.4.8 Field values do not change after parameterization and playback

Phenomena A: The corresponding field values in the records do not change after the SQL parameterization and script playback, when using Sybase ASA/SQL Anywhere database.

Solution A:

Steps to reproduce:

1. Record in the Test Mode: **DataWindow** > **Updating** > **Change Employee Salary** function.

```
"UpDateSQL_1 = update \"employee\" set \"salary\" = ? where \"emp_id\" =?……"
```

2. In the record scripts, create a new parameter "Salary" for the incremental change, and parameterize the SQL statements, as shown below:

```
"UpDateSQL_1 = update \"employee\" set \"salary\" = {salary} where
 \"emp_id\" = 105……"
```

3. Play back the scripts.

Phenomena B: When open another URL, the value of salary in records such as \"emp_id\" = 105 are found unchanged.

Solution B: The connection parameters (DisableBind=1) should be set in PowerBuilder for the Sybase ASA/SQL Anywhere database, then deploy the application source code and record the scripts again.

```
ConnectString='DSN=AppeonSample;UID=dba;PWD=sql',DisableBind=1
```

4.3.4.9 Runtime errors causing scenario failure

For the Appeon .NET version, after errors occur at runtime, the scenario cannot run correctly as before.

Phenomena: The following errors are reported in the server log:

```
2006-08-17 14:41:36.609 : Object reference not set to an instance of an object.
at Appeon.Server.Commands.CommitCommand.Execute()
at Appeon.Server.Commands.Command.ConstructAndExecute(CommandData commandData)
at Appeon.Server.Services.Controller.JRouterImpl.ExecuteInTransaction
(CommandData commandData)
at Appeon.Server.Services.Controller.JRouterImpl.ExecuteCommand
(HttpResponse response, CommandData superCommand, SessionAccessInfo session)
at Appeon.Server.Services.Controller.JRouterImpl.Call(CommandData
superCommand, HttpRequest request, HttpResponse response)
at Appeon.Server.Services.Controller.JRouterImpl.Call(Int32
clientType, Object requestData, Hashtable clientContext)
```

LoadRunner will report the following error:

```
Action1.c(190): Error: Page download timeout (120 seconds) has expired.
```

Solution: Restart IIS, SQL Server, and LoadRunner.

Check the PowerBuilder source code and change autocommit=false to autocommit=true, or move the **Commit** operation after the **Retrieve** operation, so as to ensure the commit operation will be executed for every retrieval.

4.3.4.10 Transactions failed

Many transactions in the test fail when there are large amount of concurrent access with Update or Retrieve.

Phenomena: The script playback is successful in the Virtual User Generator, and the logs in AEM report no errors. But the transaction fails when there are large amount of concurrent access.

Solution: Move the **Commit** request after the **Update** request, that is, move the **Commit** request in recorded scripts from end.c to action.c.

4.3.4.11 Unable to connect to remote servers

Phenomena: LoadRunner is not able to generate the scripts after recording and the following error appears in the log:

```
(Service=) NOT PROXIED! (REASON: Unable to connect to remote server: rc = -1 , le = 0)
```

Solution: Go to **Tools** > **Recording Options** and look at the **Port Mapping**.

Select **Socket level and WinINet level data** for **Capture level** as shown in the following figure.

Figure 4.44: Port Mapping
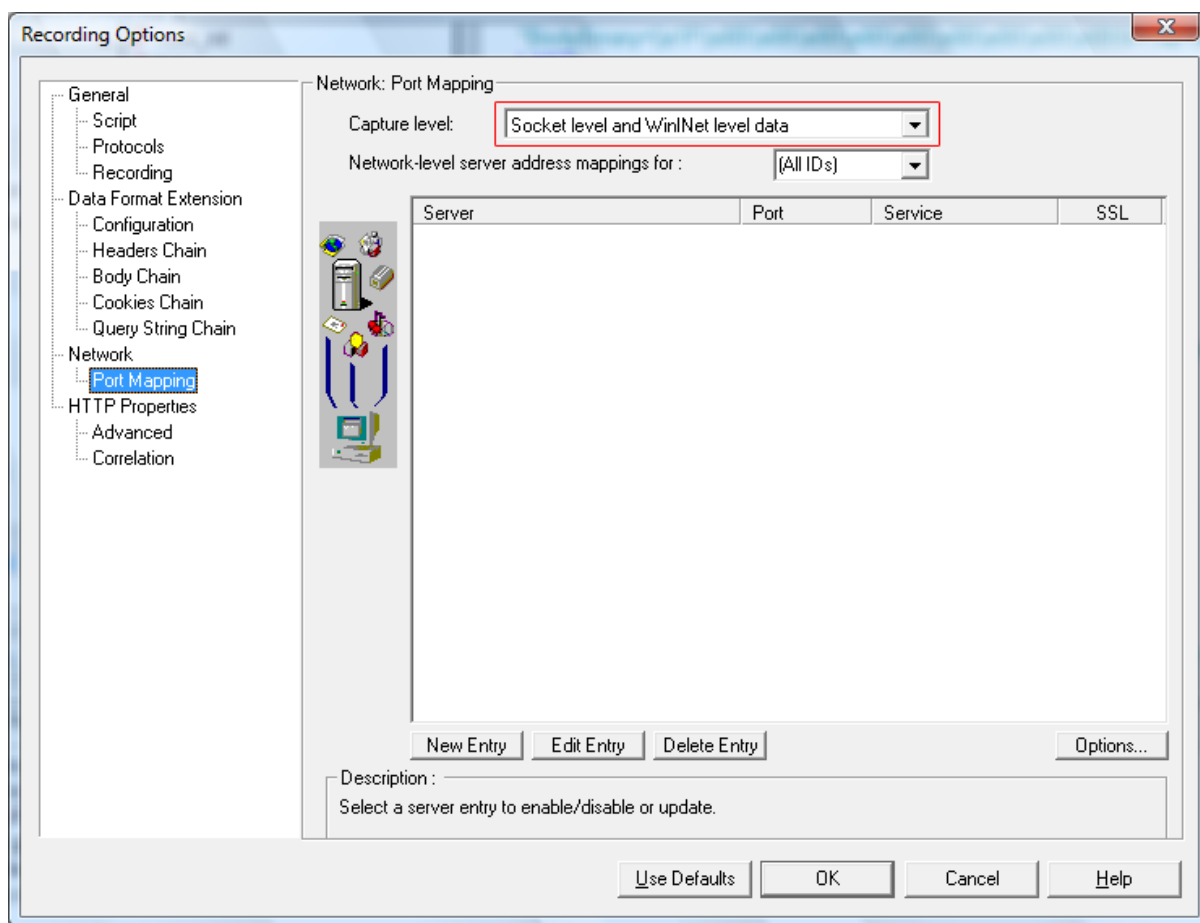
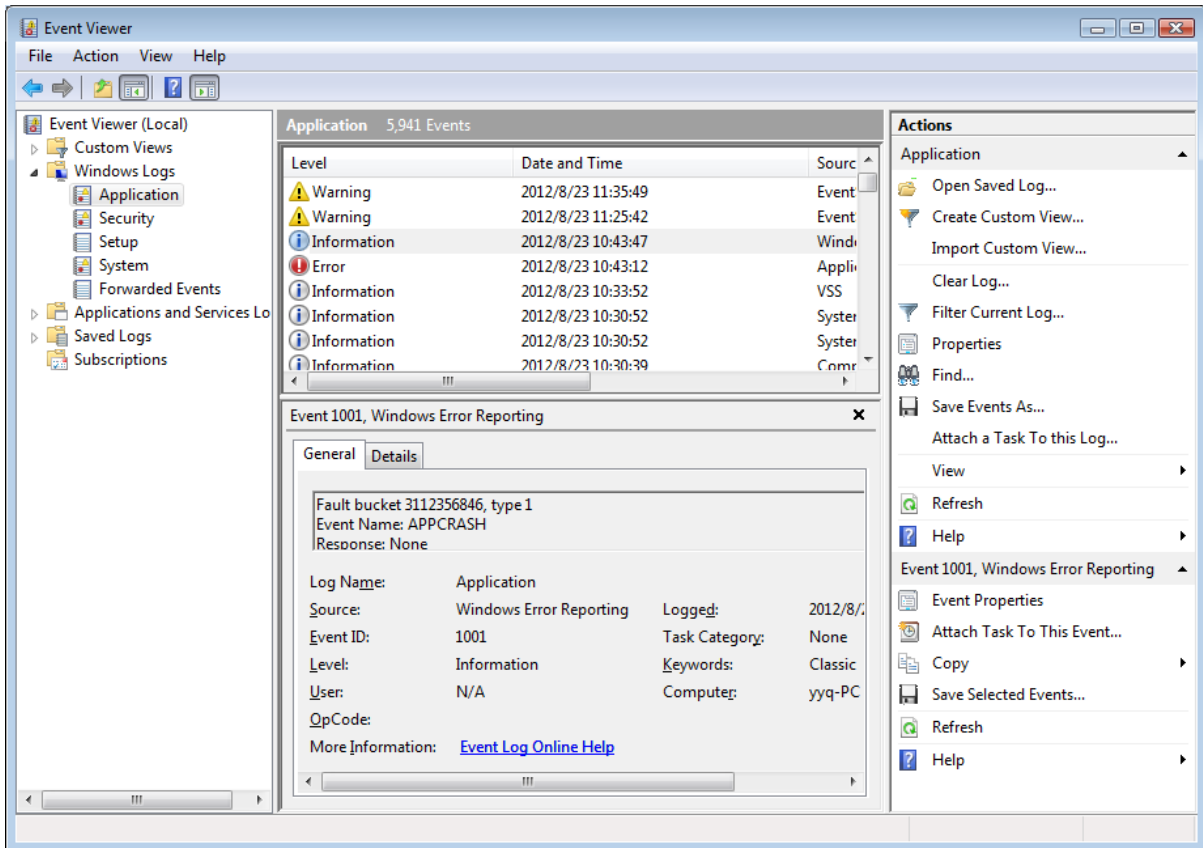## 4.4 Analyzing log files

4.4.1 Analyzing Windows application log files

Check whether there are log files about the application server in the Event Viewer, then go to the application server web site to get the detailed solution to solve those Errors.

Figure 4.45: Event Viewer

4.4.2 Analyzing Appeon Server log files

Step 1: Select Debug Mode as the Log Mode.

Figure 4.46: Log mode
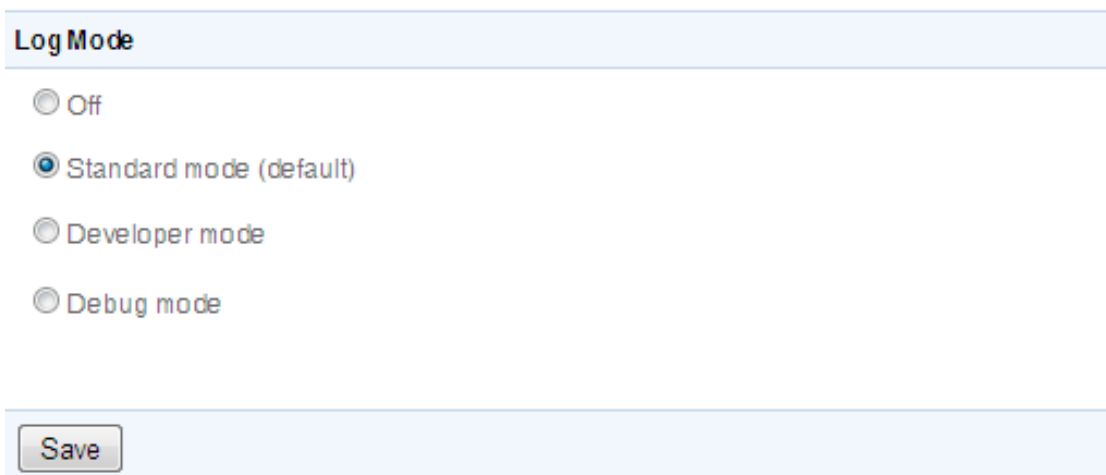
Step 2: View the AppeonServer and AppeonError.log.

Figure 4.47: Viewing logs

Read AppeonServer.log and find out operations which are time consuming and redundant.

4.4.3 Analyzing active transaction log

Step 1: Go to Active Transactions screen and select the intended Transaction ID.

Figure 4.48: Active Transactions

Step 2: Go to Active Transaction Log, you can view more details about what this transaction acted on the database recently.

Figure 4.49: Active Transaction Log

Check to view the detailed information of the transaction, maybe you can figure out which SQL statements is blocked and then optimize you PB code for Appeon.

4.5 Identifying Performance Bottlenecks of Web Server and Application Server

You can use tools like Load Runner to identify the performance bottlenecks of your web server and application server. Usually you should consider checking these places: server runtime environment, network connectivity conditions, and performance relevant parameter settings (such as timeout settings and cache settings).

Or you can refer to the information provided by your web server and application server vendor, it will help you more accurately locate the performance bottlenecks and make appropriate settings according to your actual workload.

4.6 Identifying Performance Bottlenecks of DB Server

4.6.1 Deadlock analysis

If you are doing the stress test and you find some problems during the test, you should check if there are deadlock of the database before you quite the stress test mode. By the way, if you find the performance problem when doing the database related operation, you should also check the deadlock in database.

Take SQL Server database as an example, if you want to check whether there are deadlocks or not, you can use the following way:

- Execute sp_lock in SQL query analyzer. If you find there are 'X' or 'IX' of the Mode field, then it means there are deadlocks. You can find out the deadlocks occurred in which tables by the information in Resource field.

- Execute sp_who spid in SQL query analyzer, then you can find out the host name of spid and commands being executed.

4.7 Identifying Performance Bottlenecks of PB application

If your PB application includes poorly written SQL and extensive use of non-datawindow based (direct SQL statements) SELECTs, UPDATEs, DELETEs and INSERTs, of course your application runs slow. Tune the performance of PB application is the last but the most important step. Usually the performance can be obviously improved after modifying the codes of the PB application.

4.7.1 Analyzing performance bottlenecks of PB application

Output time log for the code and find out time-consuming ones during execution. Please refer to the following Scripts and load it into PowerBuilder Application. A log with time interval regarding code will be generated while the object of_log(string arg) is called.

Please check the start time and end time of the code and find out time-consuming ones.

```
string  is_LogFile="C:\debug\logservice.log"
string  is_workdir="C:\debug\"
long    il_LogFileHandle
IF NOT directoryexists(is_workdir) THEN
 createdirectory(is_workdir)
END IF
il_LogFileHandle = FileOpen(is_LogFile,StreamMode!,write!,shared!,append!)
FileWrite(il_LogFileHandle,"~r~n~r~n")
of_log("==========Log service initialized=========")

public function integer of_log (string as_message);
IF FileExists(is_LogFile) THEN
 IF FileLength(is_LogFile) > 2097152 THEN//2M
  FileClose(il_logfilehandle)
  FileCopy(is_LogFile,is_LogFile+".bak",true)
  FileDelete(is_LogFile)
  il_LogFileHandle = FileOpen(is_LogFile,StreamMode!,write!,shared!,append!)
  END IF
END IF
as_message = "~r~n"+string(now(),"yyyy-mm-dd hh:mm:ss.fff") +as_message
FileWrite(il_logfilehandle,as_message)
return 1
end function
```

5 Tuning: DB Server

5.1 Database

The database performance will be improved from the following aspects:

- Choose proper database driver. Refer to *Recommended database driver* in *Database performance settings* for details.

- Set reasonable timeout settings. Refer to *Command Timeout* in *Recommended database setting* for details.

- Optimize the table structure.

- Use proper index.

- Optimize SQL statements.

6 Tuning: Excessive Server Calls

6.1 Overview

Excessive server calls in a given operation can create performance issues for that operation on slow and high-latency networks. If you are not familiar with the concept of "server calls", please refer to Impact of the Internet and slow networks and then proceed with this section.

This section will provide four different techniques including code examples to minimize server calls and thereby optimize the performance of your PowerBuilder application for a WAN or the Internet.

1. Partition transactions utilizing stored procedures

2. Partition non-visual logic utilizing server-side non-visual objects (NVOs)

3. Eliminating Recursive Embedded SQL

4. Group multiple server calls into one "group" call with Appeon Labels

6.2 Technique #1: partitioning transactions via stored procedures

Imagine your PowerBuilder client contains the following code:

```
long ll_rows, i
decimal ldec_price, ldec_qty, ldec_amount

ll_rows = dw_1.retrieve(arg_orderid)
for i = 1 to ll_rows
    dw_1.SetItem(i, "price", dw_1.GetItemDecimal(i, "price")*1.2)
next

if dw_1.update() < 0 then
        rollback;
        return
end if

for i = 1 to ll_rows
 ldec_price = dw_1.GetItemDecimal(i, "price")
 ldec_qty = dw_1.GetItemDecimal(i, "qty")

 if ldec_price >= 100 then
  ldec_amount = ldec_amount + ldec_price*ldec_qty
 end if
Next

ll_rows = dw_2.Retrieve(arg_orderid)
dw_2.SetItem(dw_2.GetRow(), "amount", ldec_amount)

If dw_2.update() = 1 then
 Commit;
else
 rollback;
end if
```

This is not only problematic from a runtime performance perspective since there would be numerous server calls over the WAN, but also it could result in a "long transaction" that would tie up the database resulting in poor database scalability.

The business logic and the data access logic (for saving data) are intermingled. When the first "Update( )" is submitted to the database, the related table in the database will be locked until the entire transaction is ended by the "Commit( )". The longer a transaction is the longer other clients must wait, resulting in fewer transactions per unit of time.

To improve the performance and scalability of the application, the above code can be partitioned in two steps:

1. First, move the business logic (or as much possible) outside of the transaction. In other words, the business logic should appear either before all Updates of the transaction or after Commit of the transaction. This way the transaction is not tied up while the business logic is executing.

2. Second, partition the transaction whereby all the Updates are moved into a stored procedure. The stored procedure will be executed on the database side and only return the final result. This would eliminate the multiple server calls from the multiple updates to just one server call over the WAN for saving all the data in one shot.

It is generally best to actually divide the original transaction into three segments or procedures: "Retrieve Data", "Calculate" (time-consuming logic), and "Save Data". The "Retrieve Data" procedure retrieves all required data for the calculation. This data usually would be cached in a DataWindow(s) or a DataStore(s). In the "Calculate" procedure, the data cached in DataStore will be used to perform the calculation instead of retrieving data directly from the database. The calculation result would be cached back to a DataStore and then saved to the database by the "Save Data" procedure.

Example of the new PB client code partitioned into three segments and invoking a stored procedure to perform the Updates:

```
long ll_rows, i
decimal ldec_price, ldec_qty, ldec_amount
//Retrieve data
dw_2.Retrieve(arg_orderid)
ll_rows = dw_1.retrieve(arg_orderid)
//Calculate (time-consuming logic)
for i = 1 to ll_rows
    dw_1.SetItem(i, "price", dw_1.GetItemDecimal(i, "price")*1.2)
next

for i = 1 to ll_rows
 ldec_price = dw_1.GetItemDecimal(i, "price")
 ldec_qty = dw_1.GetItemDecimal(i, "qty")

 if ldec_price >= 100 then
  ldec_amount = ldec_amount + ldec_price*ldec_qty
 end if
Next

dw_2.SetItem(dw_2.GetRow(), "amount", ldec_amount)
//Save data
declare UpdateOrder procedure for up_UpdateOrder @OrderID = :arg_orderid,
@amount = :ldec_amount;
execute UpdateOrder;
```

Example of code for the stored procedure to Update the database:

```
create procedure up_UpdateOrder(
@orderid integer,
@amount decimal(18, 2)
)
as
begin
update order_detail set price = price*1.2
where ordered = @orderid

if @@error <> 0
begin
    rollback
        return dba.uf_raiseerror()
end

update orders set amount = @amount
where ordered = @orderid

if @@error <> 0
begin
    rollback
        return dba.uf_raiseerror()
end

commit
end
```

In summary, with the above performance optimization technique, the performance and scalability is improved since the transaction is shorter. The server call-inducing Updates are all implemented on the server-side rather than the client-side, improving the response time. Secondly, moving the business logic out of the transaction further shortens the transaction. If the business logic cannot be moved out of the transaction, one may want to consider implementing the business logic together with the transaction as a stored procedure. In summary, shorter transactions equals better scalability and faster performance.

6.3 Technique #2: partitioning non-visual logic via NVOs

Partitioning non-visual logic and encapsulating it within PowerBuilder NVOs has been a long-time best practice among PowerBuilder developers. What's relatively new, however, is utilizing middleware or application server, such as SAP Sybase EAServer, Microsoft IIS, IBM WebSphere*, Oracle WebLogic*, TmaxSoft JEUS*, or JBoss* to deploy these NVOs to the server (i.e. server-side NVOs) and invoke them from the client over IIOP or HTTP.

Requires an SAP Sybase plug-in that is purchased separately from SAP or authorized resellers.

To give you a better idea of what this looks like, the following diagram shows a very high-level architecture of PowerBuilder applications utilizing server-side NVOs when deployed as a Windows Client/Server application as well as an n-tier Web (.NET* or J2EE) application with Appeon for PowerBuilder.

Figure 6.1: Architecture with server-side NVOs
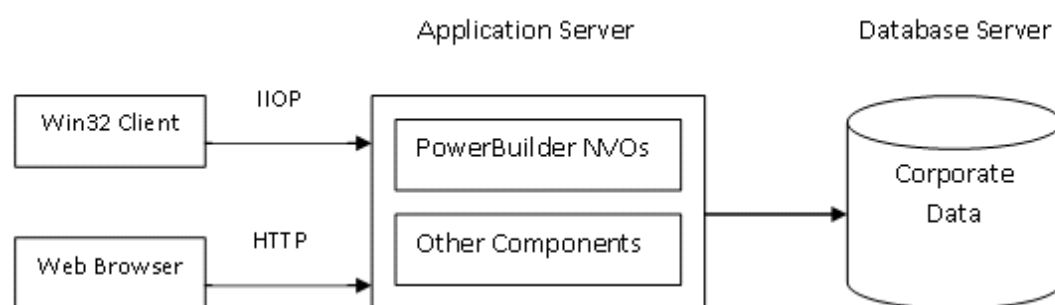
Deploying your non-visual logic as server-side NVOs is an excellent way to boost the performance of your application over a WAN; however, some thought must be given as what to partition. On one hand, consolidating your business logic within NVOs will significantly thin out client-side processing and move all those server call-inducing statements to the server-side running in a low-latency LAN environment. On the other hand, each invocation of the server-side NVO is a server call in itself.

If the non-visual logic you are partitioning contains multiple statements that result in server calls, then this would be a good candidate. However, if your non-visual logic does not contain any statements that would result in server calls, then by partitioning this you have not only created more work for yourself but actually added an additional server call that didn't exist before. So unfortunately it's not as simple as moving all non-visual logic to the server-side.

Imagine your PowerBuilder client contains the following code:

```
long ll_id

dw_1.Retrieve()
dw_1.SetSort("#1 A, #2 D")
dw_1.Sort()

declare order_detail cursor for
select id from order_detail where orderid = :arg_orderid;
open order_detail;
fetch order_detail into :ll_id;

do while sqlca.sqlcode = 0
 update order_detail set price = price*1.2
 where orderid = :arg_orderid and id = :ll_id;

 if sqlca.sqlcode < 0 then
  rollback;
  return
 end if

 fetch order_detail into :ll_id;
loop
close order_detail;
commit;

dw_2.Retrieve()
dw_2.SetFilter("price >= 100")
dw_2.Filter()
```

The code in bold above would be good candidate for partitioning as server-side NVOs while the rest of the code should remain at the PB client. After partitioning this logic, the new PB client code, which would invoke the server-side NVO, would be as follows:

```
n_order ln_order
long ll_rc

dw_1.Retrieve()
dw_1.SetSort("#1 A, #2 D")
dw_1.Sort()

ll_rc = myconnect.CreateInstance(ln_order, "PB_pkg_1/n_order")
if ll_rc = 0 then
    ln_order.of_UpdateOrderPrice(arg_orderid)
end if

dw_2.Retrieve()
dw_2.SetFilter("price >= 100")
dw_2.Filter()
```

With this technique we have reduced those numerous server calls of the database transaction to just one single call to the NVO, and at the same time created a re-usable component that can be shared by other modules in our PowerBuilder application or shared by other applications.

6.4 Technique #3: eliminating recursive Embedded SQL

It's actually quite common to find Embedded SQL in a loop, especially Select and Insert statements. As explained previously, server calls that are recursive in nature are quite dangerous, potentially generating tremendous number of server calls. If your application requires loops or recursive functions, it would be best to replace any code resulting in server calls with code that does not.

For this technique, we will assume we have Select and Insert SQL statements in a loop. The general idea is to first create a DataWindow/DataStore using the SQL. Then replace the SQL statements contained in the loop with PowerScript modifying the DataWindow/DataStore, which does not result in server calls. If the SQL statement contained in the loop is an Insert statement, we would want to replace that with PowerScript that would insert data into the DataWindow/DataStore. Once all the data has been inserted, then in one shot we would update the DataWindow/DataStore to the database (outside the loop), resulting in only one server call. If the SQL statement contained in the loop is a Select statement, we would retrieve data into a DataWindow/DataStore before executing the loop, and then write PowerScript in the loop to select the desired data from the DataWindow/DataStore.

The following is a code example that increases the price of a specific order by 20%, where Embedded SQL is used to update the change row-by-row (hence the loop), and then save those changes to the database:

```
long ll_id

declare order_detail cursor for
select id from order_detail where orderid = :arg_orderid;
open order_detail;
fetch order_detail into :ll_id;

do while sqlca.sqlcode = 0
 update order_detail set price = price*1.2
 where orderid = :arg_orderid and id = :ll_id;

 if sqlca.sqlcode < 0 then
  rollback;
  return
 end if

 fetch order_detail into :ll_id;
loop
close order_detail;
commit;
```

Now we will replace the Embedded SQL with a DataWindow. Specifically, we will cache the data in a DataWindow and update the database with a single DataWindow Update, resulting in just once server call:

```
long ll_rows, i

ll_rows = dw_1.retrieve(arg_orderid)
for i = 1 to ll_rows
    dw_1.SetItem(i, "price", dw_1.GetItemDecimal(i, "price")*1.2)
next

if dw_1.update() = 1 then
commit;
else
rollback;
end if
```

With this technique we have just eliminated server calls from inside the loop, reduced the number of server calls to just one, and created a data caching mechanism at the client-side that can be used to feed data to other controls of the PowerBuilder client.

6.5 Technique #4: grouping multiple server calls with Appeon Labels

Part of this section, mainly including introduction to Appeon Labels and the two examples, are quoted from an article titled *Appeon Performance Tuning* published in *ISUG Journal - October 2011 Edition* written by *Yakov Werde*.

Appeon deployment is a multiphase process. In the first phase PowerScript code is analyzed and converted into two categories of code (1) HTML and JavaScript that interact with the Browser and (2) JavaScript that interacts with the Appeon ActiveX component. During this phase the deployment tool converts embedded SQL and DataWindow database centric code (retrieves and updates) into RPCs (remote procedure calls) that interact with Appeon components in the Application Server. The important fact to understand is that both the Appeon deployment and runtime engines distinguish between the different categories of code. Additionally, Appeon engineers, recognizing the importance of performance tuning RPC code, provided language constructs that allow an application developer to demarcate

and group RPC code in ways that will allow the runtime engine to performance enhance browser/server communication. These performance enhancing language extensions are called Appeon Labels.[1]

Appeon for PowerBuilder provides seven "Appeon Label" functions, which can be found in the appeon_nvo_db_update object in appeon_workaround.pbl (Although the PBL is named workarounds, it also contains many useful utility classes and methods). When you examine the code of these functions, you will find the code either looks like the same nested PowerScript you have already written (such as of_update( )), or has no implementation (such as of_startqueue( )). The fact is: Appeon Label functions serve as markers (hence, the name Label) to the deployment tool; and during the deployment process, Appeon generates efficient JavaScript ActiveX component call code that implements bandwidth saving functionality in the Web tier client. Therefore, Appeon Labels do not execute any code or modify how your PowerBuilder application works in a Client/Server environment. Rather, when used on the Web it will notify Appeon's runtime Web libraries to handle certain database operations differently than PowerBuilder with the aim of reducing the number of server calls.

Below are descriptions of the seven "Label" functions about how they handle the database operation. For details about the syntax and return values of these functions, please refer to Section 2.3.6, "Appeon Labels" in *Workarounds & API Guide*.

Table 6.1: Appeon Label functions

Label	Function	Description
Commit/ Rollback Label	of_autocommitrollback	Notifies the Appeon Web application to automatically commit or roll back the first database operation statement after the label.
Commit Label	of_autocommit	Notifies the Appeon Web application to automatically commit the first database operation.
Rollback Label	of_autorollback	Notifies the Appeon Web application to automatically roll back the first database operation statement if the operation fails.
Queue Labels (Consists of Start Queue Label and Commit Queue Label)	of_startqueue of_commitqueue	These two labels must be used in pairs. They notify the Appeon Web application not to commit database operations after the Start Queue Label until the Commit Queue Label is called (and unless an Appeon Immediate Call Label is called).
Immediate Call	of_imdcall	Notifies the Appeon Web application to immediately commit a database operation.
Update Label	of_update	It is used to reduce the number of interactions with the server caused by "interrelated updates". "Interrelated updates" usually occurs when the

_____

[1]Quoted from an article titled *Appeon Performance Tuning* published in *ISUG Journal - October 2011 Edition* written by *Yakov Werde*.

Label	Function	Description
		update result of one DataWindow determines whether another DataWindow should be updated.

Appeon Commit/Rollback Label

The Appeon Commit/Rollback Label (of_autocommitrollback) notifies Appeon to automatically commit or roll back operations to the database after updating or inserting.

For example:

```
gnv_appeonDbLabel.of_AutoCommitRollback()
update tab_a ......
if sqlca.sqlcode=0 then
 ...... //code independent of database opertaions
 commit;
 ......
else
 ......
 rollback;
 ......
endif
```

Appeon Commit Label

The Appeon Commit Label (of_autocommit) notifies Appeon to commit operations to database after updating and inserting.

For example:

```
gnv_appeonDbLabel.of_AutoCommit()
update tab_a ......
```

Appeon Queue Labels

There are two Appeon Queue Labels, the Start Queue Label (of_startqueue) and the Commit Queue Label (of_commitqueue). These two labels must be used in pairs. They notify Appeon not to commit database operations after the Start Queue Label until the Commit Queue Label is called (and unless an Appeon Immediate Call Label is called).

For example:

```
gnv_appeonDbLabel.of_StartQueue()
dw_1.retrieve(arg1,arg2)
dw_2.retrieve(arg3,arg2)
......
dw_3.retrieve(arg4)
gnv_appeonDbLabel.of_CommitQueue()
```

Appeon Immediate Call Label

The Appeon Immediate Call Label (of_imdcall) is used between the Appeon Start Queue Label and Appeon Commit Queue Label, when the return value of an operation that is called after the Appeon Start Queue Label determines the subsequent business logic, for example, the return value is used in a CASE or IF...THEN expression.

For example:

```
gnv_appeonDbLabel.of_StartQueue()
dw_1.retrieve()
gnv_appeonDbLabel.of_ImdCall()
select ... into :var_1,:var_2 ......

if var_1>0 then
 para = "ok"
else
 para = "false"
end if
dw_2.retrieve(para)
gnv_appeonDbLabel.of_CommitQueue()
```

Appeon Update Label

The Appeon Update Label (of_update) is used to reduce the number of interactions with the server caused by "interrelated updates". "Interrelated updates" usually occurs when the update result of one DataWindow determines whether another DataWindow should be updated.

The following example shows how Appeon uses the Update Label to reduce client-server interactions:

Example of interrelated updates:

```
if dw_1.Update()=1 then
    if dw_2.Update()=1 then
        commit;
        Messagebox("Success","Update success!")
    else
        rollback;
        Messagebox("Failure","Update all failure!")
    end if
else
    rollback;
    Messagebox("Failure","Update dw_1 failure!")
End if
```

Use the Appeon Update Label to rewrite the example:

```
l_rtn = gnv_appeonDb.of_Update(dw_1,dw_2)
if l_rtn=1 then
   Messagebox("Success","Update success!")
elseif l_rtn= -102 then
   Messagebox("Failure","Update all failure!")
Else
    Messagebox("Failure","Update dw_1 failure!")
End if
```

Script defined in the Update Label associated function, of_Update(dw_1,dw_2):

```
if dw_1.Update()=1 then
    if dw_2.Update()=1 then
        commit;
    return 1
    else
        rollback;
    return -102
    end if
else
rollback;
return -101
end if
```

The more database operations utilize Appeon Labels, the faster the performance will be. For PowerBuilder applications deployed to the Web with Appeon for PowerBuilder, in many cases you will achieve acceptable runtime performance simply by utilizing this technique. The reason is that there are a number of features built into Appeon's infrastructure framework that automatically boost the performance of PowerBuilder applications over the Web. The performance boosting features are discussed in Automatic performance boosting.

Two Examples[1]

Now, let's see two situations where Appeon Labels reduce network traffic.

The first example illustrates performance enhancements when chaining DataWindow updates. Oftentimes, data contained in multiple data controls (datawindow, datastore or childdatawindow) must be grouped into a logical unit of work (LUW).

The following figure shows the pseudocode update algorithm. This code necessitates three browser to server round trips, one for each data control.

Figure 6.2: Nested Update Algorithm

```
IF dw_1.update( true, false) then
    IF ds_1.update( true, false) then
        IF dwc_1.update( ) then
            commit;
        ELSE
            rollback;  //dwc failed
    ELSE
        rollback;    //ds failed
ELSE
    rollback;        //dw failed
END IF
```

(Appeon groups all changed data into a single transmission). As you can see from the following figure, refactoring the code to use the Appeon Label of_update( ) method, reduces three round trips one. In the event of a failed update the method returns a result code indicating which update failed.
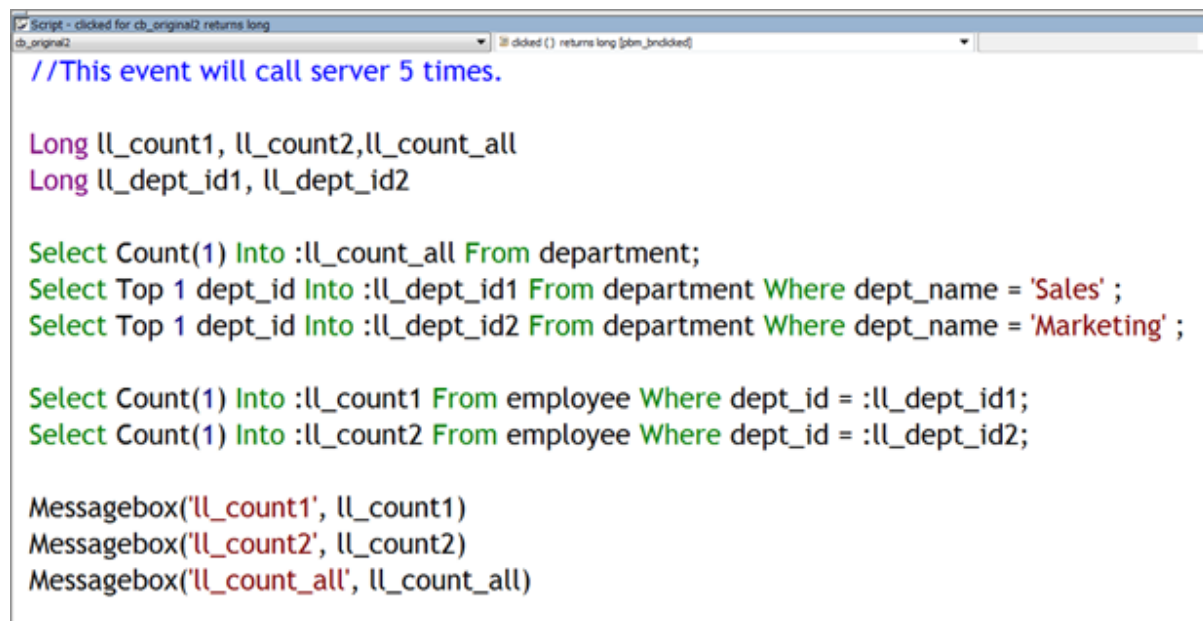
Figure 6.3: Call Appeon of_update method( )

```
//This event will call server 1 time.

dw_1.AcceptText()
dw_2.AcceptText()
dw_3.AcceptText()

gnv_appeondb.of_update( dw_1, dw_2, dw_3)
```

Sometimes a script has multiple embedded SQL statements grouped into a single LUW. Each SQL statement causes a browser to server round trip. The following figure illustrates one such (simplified) scenario.

Figure 6.4: Five Round Trips

```
Script - clicked for cb_original2 returns long
cb_original2                                        clicked ( ) returns long [pbm_bnclicked]

//This event will call server 5 times.

Long ll_count1, ll_count2,ll_count_all
Long ll_dept_id1, ll_dept_id2

Select Count(1) Into :ll_count_all From department;
Select Top 1 dept_id Into :ll_dept_id1 From department Where dept_name = 'Sales' ;
Select Top 1 dept_id Into :ll_dept_id2 From department Where dept_name = 'Marketing' ;

Select Count(1) Into :ll_count1 From employee Where dept_id = :ll_dept_id1;
Select Count(1) Into :ll_count2 From employee Where dept_id = :ll_dept_id2;

Messagebox('ll_count1', ll_count1)
Messagebox('ll_count2', ll_count2)
Messagebox('ll_count_all', ll_count_all)
```
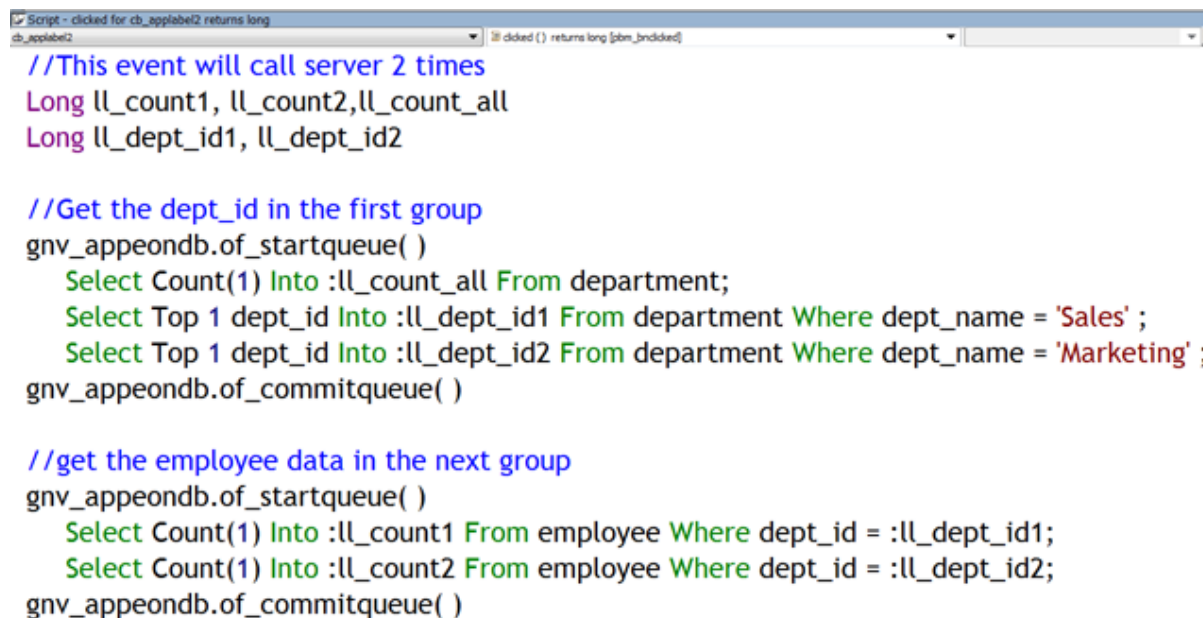
In this case you can use the Appeon Labels of_startqueue( ) and of_endqueue( ) to demarcate logical statement groupings thereby reducing the number of server round trips. The code shown in the following figure reduces the number of server round trips by over 50% by dividing the SQL into two logical groups; the first group of statements acquires values necessary for statements in the second group.

Figure 6.5: Reduced to 2 Round Trips

```
Script - clicked for cb_applabel2 returns long
cb_applabel2                                        clicked ( ) returns long [pbm_bnclicked]

//This event will call server 2 times
Long ll_count1, ll_count2,ll_count_all
Long ll_dept_id1, ll_dept_id2

//Get the dept_id in the first group
gnv_appeondb.of_startqueue( )
    Select Count(1) Into :ll_count_all From department;
    Select Top 1 dept_id Into :ll_dept_id1 From department Where dept_name = 'Sales' ;
    Select Top 1 dept_id Into :ll_dept_id2 From department Where dept_name = 'Marketing' ;
gnv_appeondb.of_commitqueue( )

//get the employee data in the next group
gnv_appeondb.of_startqueue( )
    Select Count(1) Into :ll_count1 From employee Where dept_id = :ll_dept_id1;
    Select Count(1) Into :ll_count2 From employee Where dept_id = :ll_dept_id2;
gnv_appeondb.of_commitqueue( )
```

7 Tuning: Heavy Client

7.1 Overview

If you find your Appeon Web application performs poorly even when run in a local environment, then chances are the JavaScript interpreter of the Web browser is slowing you down. Generally, a newer or more high-power computer will clear up many situations. But if it is not an option to utilize a late model computer or if the performance issue still persists, then you would want to consider optimizing your PowerBuilder code to make it more efficient when running in the Web browser. This section provides several different techniques for dealing with several specific types of inefficient PowerBuilder coding practices.

7.2 Technique #1: thin-out #heavy# Windows

Redesign the navigation strategy to present a lighter-weight Client user interface. Specifically, you would want to focus on reduce the number of DataWindows and DropDownDatawindows in a particular window or tab page. In many situations, the DataWindows in a Window can be spread out across multiple Windows or tabs, thereby reducing the "weight" of the Window. It may be possible to rework your DropDownDataWindows as DropDownListBoxes. By thinning out the UI, it will not only make the Window run faster but your users will not be overwhelmed by so much data.

7.3 Technique #2: thin-out #heavy# UI logic

This section is broken down into several subsections, utilizing the same technique to deal with various types of "heavy" UI logic.

7.3.1 Manipulating the UI in loops

Excessive and unnecessary loops have a negative impact on performance. Some PowerBuilder code will trigger your Appeon Web application to redraw visual objects, such as DataWindows, controls, etc. If such functions are put into a loop, it will redraw the visual objects numerous times and therefore negatively affecting performance.

Appeon recommends the following:

• Do not put functions that operate on DataWindow rows into a loop.

• Avoid placing functions that result in the repaint of visual control(s) into a loop; otherwise, the visual control(s) will be repainted many times while the loop is executed.

• Use the Find function for DataWindow search instead of using the loop statement.

The following is an example:

```
Long ll_row
String ls_expression
String ls_Name
ls_Name = "Mike"
For ll_row = 1 To dw_1.RowCount()
 ls_expression =
  dw_1.GetItemString(ll_row,"name")
 If ls_expression = ls_Name Then


  Exit
 End If
Next
```

```
Long ll_row
Long ll_rowcount
String ls_expression
String ls_Name
ls_Name = "Mike"
ll_rowcount = dw_1.RowCount()
For ll_row = 1 To ll_rowcount
 ls_expression = +
 dw_1.GetItemString(ll_row,"name")
 If ls_expression = ls_Name Then
    ...
 Exit
End If
Next
```

```
Long ll_row
Long ll_rowcount
String ls_expression
String ls_Name
ls_Name = "Mike"
ll_rowcount = dw_1.RowCount()
ls_expression = "name = '" + ls_Name + "'"
ll_row = dw_1.Find(ls_expression,1,ll_rowcount)
...
```

7.3.2 Triggering events repeatedly

Frequent triggering of events such as Timer, MouseMove, and SelectionChanging slows
down performance. For example, once a Timer event is triggered, it occurs repeatedly at
a specified interval that can be set to even milliseconds (1/1000 of a second). Minimize
the usage of events with high repetition such as Timer, MouseMove, SelectionChanging,
GetFocus, LoseFocus, Activate, and Deactivate.

7.3.3 Performing single repetitive tasks

Use batch operations instead of performing a single operation many times. For example, the
execution of the following "batch" code is two to three times faster than the original code.

```
For I = 1 To 100
 dw_1.SetItem(ll_i,+
                            "name",+
                            dw_2.GetItemString(ll_i,"name"))
                    ...
Next
```

⇩

```
dw_1.RowsCopy(1,100,Primary!,dw_2,1,100,Primary!)
```

7.3.4 Initializing #heavy# tabs

For windows containing Tab controls, if the Tab control contains more than five tab pages or the initialization of each tab page is complex, Appeon recommends you use the CreateOnDemand method of Tab control to improve the runtime performance of these windows.

When CreateOnDemand is enabled, only the current tab page that is created will be initiated. The initialization of the hidden tab pages takes place only when they are selected. Therefore the window will operate more efficiently.

Please keep in mind that as you make this change you may also need to modify other code of the Tab control. For example, if the current tab page uses data from another page, you need to:

1. Move the script that is used for obtaining data, to the SelectionChanged event.

2. Add a condition to validate whether the tab page carrying the data has been initiated. If initiated, the current tab page will successfully obtain the data; if not initiated, the user must select the tab page for initialization purposes, and the current tab page will successfully obtain the data.

7.3.5 Using ShareData or RowsCopy/RowsMove for data synchronization

The following PowerBuilder functions can synchronize data between DataWindows: ShareData, RowsCopy/RowsMove, Object.Data, and SetItem. ShareData is the fastest and it is recommended to use it whenever you need to synchronize data between DataWindows. SetItem is the slowest and should be avoided as much as possible. If ShareData cannot be or should not be used for some reason, then consider using RowsCopy/RowsMove followed by Object.Data.

7.3.6 Using computed fields

Computed columns involve a lot of recalculation in many situations; for example, when a column is deleted, added, or renamed. This recalculation is a process-intensive task, which negatively impacts performance and can be worked around. Therefore, Appeon recommends the following:

• Avoid using computed columns in detail bands. Instead, add expressions in the SQL statements for getting specific data.

• Avoid embedding a computed column in an existing computed column.

- If a computed column is "Text: Sum or Expression", it is recommended that you divide the column into two columns: an edit style column with the "Text", and a computed column with "Sum or Expression".

7.3.7 Using DataWindow expressions

Generally speaking, DataWindow expressions will slow-down the initial display or subsequent refresh of DataWindows. As such, Appeon recommend you reduce the usage of DataWindow expressions if possible, especially in the following situations:

- Avoid using DataWindow expressions for computing and setting column properties.

- Avoid setting sort and filter criteria directly for a DataWindow object. Instead, write the sort and filter criteria in the SQL statement of the DataWindow object. As noted previously, it is faster to use SQL statements than DataWindow functionality.

7.3.8 Using complex filters

Filters are considered "complex" if the filter criteria contain one or more expressions that call to one or more functions. It is recommended that you not use complex filtering on a DataWindow, especially on a DataWindow that has large amounts of data.

7.3.9 Using RowsFocusChanging/RowsFocusChanged events

The DataWindow RowsFocusChanging and RowsFocusChanged events can be triggered under many situations, especially when a DataWindow retrieves data. Since data is usually automatically retrieved into DataWindows when a Window is opened, if a lot of code is written into the RowsFocusChanging and RowsFocusChanged events, it will significantly prolong the time it takes to open the Window and display the DataWindow. Appeon recommends that you do not write code into RowsFocusChanging and RowsFocusChanged events unless it is necessary.

7.4 Technique #3: offload #heavy# non-visual logic

Instead of trying to write "heavy" logic more efficiently or avoiding use of "heavy" logic, the simplest way is just to offload all that "heavy" logic to the application server, which is designed to handle the most daunting tasks. The only catch is that only non-visual logic can be run at the application server.

The following types of non-visual logic can be encapsulated in PowerBuilder NVOs and deployed to the application server to eliminate "heavy" logic from the Web browser:

- Complex non-visual events and functions, especially non-visual events and functions that contain dynamic SQL, Cursor statements, Stored Procedure calls, and other SQL statements.

- Validation of updated data.

- A series of data computations or a complex data computation.

8 Tuning: Large Data Transmissions

This section introduces two common-used techniques to reduce the size of data transmission:

8.1 Overview

Suppose you have worked hard to make an application Web-ready using Appeon, and, using your test data, it seemed to perform acceptably. Then, when your users provide "live" test data in realistic volumes, you discover that the application takes a long time to load, and worse, a long time to respond to your user's input. What to do?

Well first you should confirm that your issue is not being caused by excessive server calls (see Tuning: Excessive Server Calls). The reason is that majority of the time, PowerBuilder applications are coded such that as additional rows of data are retrieved logic is executed to validate, manipulate, or otherwise handle the data, which can result in server calls. As such, the more rows of data are retrieved the more server calls are made.

Once you are certain the slow-down is not caused by excessive server calls then you can consider reducing the size of data transmission. So what can do practically?Well at a high-level there are several techniques you can employ:

- The first and most popular is staging the data retrieval into manageable increments. For example, you can expose a Next button, and have the application respond to this button click by getting the next logical segment of the result set just like typical Websites or Web applications. Technique #1: retrieving data incrementally gives you instructions on how to achieve this.

- Another technique is to create multiple smaller "specific" views rather than one larger "general" view. Consider adding SQL WHERE clauses based on more search criteria, thus retrieving only the amount of data that is absolutely necessary for a particular view of interest.

- If you have a choice between reducing the number of rows retrieved, and reducing the number of columns, note that a small reduction in columns (described below in Technique #2: minimizing excessive number of columns) can improve performance to an even greater extent than a reduction in rows. This is because most of the time, loops, whether in the application code or in the virtual machine, visit columns first and then rows.

Anything you do to reduce the size of the result set in one way or another can only improve performance and possibly improve usability of your application as well.

8.2 Technique #1: retrieving data incrementally

8.2.1 For Oracle database server

Oracle includes a pseudo-column called ROWNUM which allows you to generate a list of sequential numbers based on ordinal row. If your application uses Oracle database, apply your Oracle skills and ROWNUM to limit the number of returned rows. For example, this query selects the TOP 10 rows from a table:

```
SELECT *
FROM   (SELECT * FROM my_table ORDER BY col_name_1)
WHERE  ROWNUM BETWEEN 1 AND 10;
```

You can impose a NEXT button to the DataWindow. In the Clicked event of the NEXT button, the query changes with ROWNUM increments by 10. Therefore, when the NEXT button is clicked, the DataWindow displays next 10 rows.

8.2.2 For all other database servers

If your application uses a non-Oracle database (for example, Microsoft SQL server) you can use the following SQL syntax to limit the number of returned rows to the DataWindow:

```
SELECT TOP 10 *
FROM my_table
WHERE Table.primary_key > = bottom
ORDER BY Table.primary_key;
```

"bottom" is a variable that contains the row number of the first row you want to retrieve, where rows are ordered by the primary key for the table. Before retrieving the first page of data, "bottom" should be set to a value smaller than any primary key value in the table.

Based on this SQL statement, you can implement Next and Previous buttons for the DataWindow. Their Clicked events increment or decrement the bottom variable so that its value matches the primary key value in the first row you want to retrieve then execute the above SQL statement.

8.3 Technique #2: minimizing excessive number of columns

As the number of rows in the result set increased, the number of columns will cause greater degradation on performance, especially for nested loops in your application which process rows in the outer loop, and columns in the inner loop. Sometimes the excessive number of columns is intentional and other times it is unintentional.

A sign of unintentionally excessive columns would be the SQL syntax Select * From: consider modifying this syntax to Select fieldList From, where fieldList is the comma-separated list of all, and only, those fields your application will actually need. The performance of the SQL syntax using asterisk will be automatically degraded any time your database administrator modifies the database design by adding columns.

A sign of intentionally excessive columns is simply a long list of columns in your SQL Select statement. Consider analyzing your actual needs to make certain all columns are necessary. It may be possible to request certain columns (needed only in exceptional circumstances) in a separate SQL operation. Please keep in mind if the Visible property of a column is set to zero (the control is not visible), even though the Column cannot be seen, it is still impacting performance.

9 Conclusion

PowerBuilder applications that perform well today in your local network may not perform well in a distributed architecture tomorrow. Likewise, typical PowerBuilder development practices may not be suitable for a distributed architecture. The several techniques outlined in this guide are intended to steer you in general directions. It is recommended to extrapolate from these examples and apply to your particular situation. Please keep in mind that excessive server calls is the single biggest culprit of performance issues over the Internet, which is a relatively high latency connection.

Purchasing expensive network connectivity and faster hardware can make up for suboptimal code. Sometimes the cost of doing this is less than the cost of optimizing the code. If you do take this route, keep in mind that a low-latency network connection is generally the key rather than a high-bandwidth connection. Reason being, for most PowerBuilder applications and deployments, it is the network latency that kills the runtime performance not bandwidth limitations.

Index

Symbols

"The session does not exist" error, 63

A

additional functions, 38

Additional steps for the Update operation, 57, 62

advanced thread settings

  execution Timeout, 18

  maxconnection, 17

  maxWorkerThreads and maxIoThreads, 17

  minFreeThreads and minLocalRequestFreeThreads, 17

  minWorkerThreads, 17

  recommended thread settings, 18

AEM performance settings

  Custom Libraries download settings, 13

  DataWindow data caching, 13

  Log file settings, 14

  Multi-thread download settings, 13

  Timeout settings, 12

Analyzing log files

  Analyzing log files, 70

appendix

  Troubleshooting, 63

Appeon Developer performance settings, 11

Appeon Enterprise Manager performance settings, 12

Appeon performance

  automatic performance boosting, 9

  expected performance level, 9

  impact of heavy client-side logic, 7

  impact of large data transmission, 8

  impact of the Internet and slow networks, 6

Appeon performance analyzer

  additional functions, 38

  calls analysis, 26

  download analysis, 30

  enable, 21

  get to know, 24

  remove, 24

  start, 22

  system configuration, 25

  view detail, 31

automatic performance boosting, 9

C

cache size, 15

calls analysis, 26

configure data sources

  cache size, 15

  JDBC driver, 14

Configuring AEM, 49

Custom Libraries download settings, 13

D

Data preparation, 49

DataWindow data caching, 13

download analysis, 30

E

eliminate recursive Embedded SQL, 79

enable Appeon performance analyzer, 21

error message appears in script playback, 65

Error message in Appeon Log, 65

Errors appear when playing back scripts with LoadRunner 8.0, 63

excessive server calls

  eliminate recursive Embedded SQL, 79

  group multiple server calls with Appeon Labels, 80

  partition non-visual logic via NVOs, 77

  partition transactions via stored procedures, 75

execution Timeout, 18

expected performance level, 9

F

Failed to parameterize scripts, 65

Field values do not change after parameterization and playback, 68

Filter function, 38

Find function, 44

G

General Limitations on Performance Testing, 48

get to know Appeon performance analyzer, 24

group multiple server calls with Appeon Labels, 80

H

Heavy Client

  offload heavy non-visual logic, 89

other database servers, 91
Rollback Completion time, 13
Running Scenarios, 62
Runtime errors causing scenario failure, 68

S

Save As XML function, 45
Session timeout, 12
Session Timeout Detection Interval, 12
Setting Scenarios, 61
setting scenarios
   additional steps for update operation, 62
start Appeon performance analyzer, 22
Sybase EAServer performance settings
   configure data sources, 14
   JVM startup option, 14
system configuration, 25

T

testing Appeon Web applications with
LoadRunner
   general limitations, 48
   loadrunner, 48
   testing steps, 49
testing steps, 49
   modifying, 56
   playing back, 61
   preparing test cases, 50
   recording, 50
   Running scenarios, 62
   Setting scenarios, 61
Testing steps
   Configuring AEM, 49
   data preparation, 49
The value of sessionID is null, 64
thin-out "heavy" UI logic
   perform single repetitive tasks, 87
thin-out heavy UI logic, 86
   initialize heavy tabs, 88
   manipulate the UI in loops, 86
   trigger events repeatedly, 87
   use complex filters, 89
   use computed fields, 88
   use DataWindow expressions, 89
   use RowsFocusChanging/
   RowsFocusChanged events, 89
   use ShareData or RowsCopy/RowsMove
   for data synchronization, 88
thin-out heavy windows, 86

Timeout settings
   Maximum Rollback Retries, 13
   Rollback Completion time, 13
   Session timeout, 12
   Session Timeout Detection Interval, 12
   Transaction timeout, 12
   Transaction Timeout Detection Interval,
   12
Transaction timeout, 12
Transaction Timeout Detection Interval, 12
Transactions failed, 69
trigger events repeatedly, 87
Troubleshooting, 63
troubleshooting
   error message, 65
   error message in appeon log, 65
   Errors appear, 63
   Failed to parameterize scripts, 65
   Field values, 68
   Out of memory error, 67
   Runtime errors, 68
   The session, 63
   The value, 64
   Transactions failed, 69
   Unable to connect, 69
Tuning: DB server
   Database, 74

U

Unable to connect to remote servers, 69
use complex filters, 89
use computed fields, 88
use DataWindow expressions, 89
use RowsFocusChanging/
RowsFocusChanged events, 89
use ShareData or RowsCopy/RowsMove for
data synchronization, 88

V

view detail, 31

W

Web and application server performance
settings
   Microsoft IIS server, 15
   SAP Sybase EAServer, 14