Appeon Performance Tuning Guide

Appeon® 3.1 for PowerBuilder®

FOR WINDOWS

DOCUMENT ID: DC10089-01-0310-01

LAST REVISED: September 13, 2005

Appeon Corporation, 1/F, Shell Industrial Building, 12 Lee Chung Street, Chai Wan District, Hong Kong.

Contents

1 About This Book 1
1.1 Audience 1
1.2 How to use this book 1
1.3 Related documents 1
1.4 If you need help 3

2 Appeon 3.1 Performance Levels 4
2.1 Two deployment options 4
2.2 Deployment performance 4
2.2.1 Expected performance for full deployment 4
2.2.2 Expected performance for incremental deployment 5
2.2.3 Estimate full deployment time 5
2.3 Client runtime performance 5
2.3.1 Overview 5
2.3.2 Runtime performance with Appeon Pure-JavaScript 6
2.3.3 Runtime performance with Appeon Xcelerator 8
2.4 Performance impact of the Internet 9

3 Performance Tools and Methods 11
3.1 Overview 11
3.2 Adjustment to performance-related settings 11
3.2.1 Performance boosters provided in Appeon Developer 11
3.2.2 DataWindow Data Cache 12
3.2.3 Custom Libraries download settings 12
3.2.4 File caching in Internet Explorer 12
3.2.5 Performance of Appeon Server 13
3.2.6 Database performance tuning 14
3.3 Performance tuning tools 15
3.3.1 Heavy Window report 15
3.3.2 Web performance runtime tracing report 15

4 Performance Tuning Basics 18
4.1 Common causes of poor-performing applications 18
4.1.1 Excessive data retrieval 18
4.1.2 “Heavy” Windows 19
4.1.3 Large “global” overhead 19
4.1.4 Intermingled and complex business logic 20
4.2 Tuning the features that commonly degrade performance 21
4.2.1 Thin-out “heavy” Windows 21
4.2.2 Decrease “global” overhead 21
4.2.3 Consolidate business logic into NVOs 22
4.3 What to do next 22

5 Performance Fine-Tuning 23
5.1 Reducing the size of data set requests 25
5.1.1 Size of requests for data 25

5.1.2 Retrieval of an excessive number of columns 25
5.2 Simplifying application complexity 26
5.2.1 Object reference 26
5.2.2 Inheritance 26
5.2.3 Nesting level 27
5.3 PowerScript tuning 27
5.3.1 Minimizing loops 27
5.3.2 Minimizing complicated and special arrays 28
5.3.3 Using a defined standard data type instead of any data type 28
5.3.4 Reworking Dot Notation expressions with variables 29
5.3.5 Reworking calculations with variables 29
5.3.6 Avoiding sharing names among variables 30
5.3.7 Reducing global definitions 30
5.4 Events and functions 30
5.4.1 Avoiding dynamic calls 30
5.4.2 Minimizing code in Open, Resize, Active, and Constructor events 30
5.4.3 Avoiding using events that are triggered repeatedly 30
5.4.4 Using batch operations 31
5.5 Controls and objects 31
5.5.1 Reducing number of advanced controls 31
5.5.2 Reducing and optimize picture-based controls 31
5.5.3 Limiting submenu levels 31
5.5.4 Using TreeView and ListView more efficiently 31
5.5.5 Using non-autoinstantiated objects whenever possible 32
5.6 Window 32
5.6.1 Thinning out “heavy” Windows 32
5.6.2 Using OpenSheet instead of Open 32
5.6.3 Avoiding cross-Window calling or inter-Window calling 32
5.7 DataWindow 33
5.7.1 Speeding up data-retrieving process 33
5.7.2 Using ShareData or RowsCopy/RowsMove for data synchronization 34
5.7.3 Using “Order by” instead of Sort property for data sorting 34
5.7.4 Reducing the number of columns in DataWindows 35
5.7.5 Avoiding computed fields 35
5.7.6 Reworking DropDownDataWindows and reduce EditMask edit styles 35
5.7.7 Reducing usage of DataWindow expressions 35
5.7.8 Avoiding complex filtering 36
5.7.9 Using Describe and Modify to get and set DataWindow object properties 36
5.7.10 DataStore is preferred over DataWindow for non-visual data processing 36
5.7.11 Using SetRow instead of ScrollToRow 36
5.7.12 Reducing usage of DataWindow SQLPreview event 36
5.7.13 Minimizing code in RowsFocusChanging and RowsFocusChanged events 37
5.7.14 Performance tuning for Composite DataWindows 37
5.8 Reducing client-server interaction 37

Index 39

1 About This Book

1.1 Audience

This book is intended to help 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 with Appeon 3.1. It is also intended to guide PowerBuilder architects and developers on how to build PowerBuilder applications that perform well when deployed to the Web.

1.2 How to use this book

There are five chapters in this book:

Chapter 1: About This Book

A general description of the contents of this document.

Chapter 2: Appeon 3.1 Performance Levels

Describes current deployment/runtime performance levels of Appeon 3.1.

Chapter 3: Performance Tools and Methods

Describes the main two kinds of performance tools and methods that can assist you in improving the performance of deployed applications.

Chapter 4: Performance tuning Basics

Introduces basic performance tuning concepts to achieve good Web performance, such as thinning “heavy” Windows and decreasing “global” overhead.

Chapter 5: Performance Fine-tuning

Documents in detail PowerBuilder features and coding styles that result in poor Web performance. Provides workarounds for these features and general guidelines that will result in good Web performance.

1.3 Related documents

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

- *Appeon Demo Applications Tutorial*:

Introduces Appeon’s demo applications, including the Appeon Sales Application Demo, Appeon Code Examples, and the Appeon ACF Demo, which show Appeon’s capability in converting PowerBuilder applications to the Web.

- *Appeon Developer User Guide* (or *Working with Appeon Developer Toolbar*)

Provides instructions on how to use the Appeon Developer toolbar in Appeon 3.1.

Working with Appeon Developer Toolbar is an HTML version of the *Appeon Developer User Guide*.

- *Appeon Enterprise Manager User Guide*:

Introduces the Appeon Enterprise Manager, a Web application that maintains Appeon Web

applications and Appeon Server over the Internet, an intranet, or an extranet.

- *Appeon Supported Features Guide for Appeon Xcelerator* (or *Appeon Features Help for Appeon Xcelerator*):

Provides a detailed list of what PowerBuilder features are supported and can be converted to the Web with Appeon 3.1, using the Appeon Xcelerator deployment option, and what features are unsupported.

Appeon Features Help for Appeon Xcelerator is an HTML version of the *Appeon Supported Features Guide for Appeon Xcelerator Deployment*.

- *Appeon Supported Features Guide for Pure-JavaScript* (or *Appeon Features Help for Pure-JavaScript*):

Provides a detailed list of what PowerBuilder features are supported and can be converted to the Web with Appeon 3.1, using the Pure-JavaScript deployment option, and what features are unsupported.

Appeon Features Help for Pure-JavaScript is an HTML version of the *Appeon Supported Features Guide for Pure-JavaScript Deployment*.

- *Appeon Installation Guide*:

Provides instructions on how to install *Appeon for PowerBuilder* successfully.

- *Appeon Migration Guide*:

A process-oriented guide that illustrates the complete diagram of the Appeon Web migration procedure, and includes various topics related to steps in the procedure.

- *Appeon Migration Tutorial*:

A tutorial that walks the user through the entire process of deploying a small PowerBuilder application to the Web.

- *Appeon Performance Tuning Guide*:

Provides instructions on how to modify a PowerBuilder application to achieve better performance with its *corresponding Web application.*

- *Appeon Troubleshooting Guide*:

Provides information about troubleshooting issues, covering topics such as product installation, Web deployment, AEM, Web application runtime, etc.

- *Introduction to Appeon*:

Guides you through all the documents included in Appeon 3.1 for PowerBuilder.

- *Using the PowerBuilder Foundation Class Library with Appeon* (or *Appeon-compliant Framework Reference*):

Provides a detailed list of what PowerBuilder PFC features are supported and can be converted to the Web with Appeon, and what features are not supported.

Appeon-compliant Framework Reference is an HTML version of the *Using the PowerBuilder Foundation Class Library with Appeon*.

- *What's New in Appeon*:

Introduces new features and changes in Appeon 3.1 for PowerBuilder.

1.4 If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support or an Authorized Sybase Support Partner. If you have any questions about this product or if you need assistance during the installation process, ask the designated person to contact Sybase Technical Support or an Authorized Sybase Support Partner based on your support contract. You may access the Technical Support Web site at http://www.sybase.com/support.

2 Appeon 3.1 Performance Levels

2.1 Two deployment options

Appeon 3.1 provides two deployment options, Pure-JavaScript deployment and Appeon Xcelerator deployment.

- The Pure-JavaScript deployment is unchanged from that in Appeon 2.8 and 3.0.

  Pure-JavaScript deployment can support large applications up to 100 megabytes in size.

  Just as in Appeon 2.8 and 3.0, the Pure-JavaScript deployment time is on par with PowerBuilder deployment to EXE. The runtime client performance of Pure-JavaScript applications, when run in a 100Mbps internal network, follows a "fewer-than-10-DataWindows/fewer-than-5-second" rule.

- The Appeon Xcelerator deployment has been enhanced by 10% to 50% in both deployment performance and runtime performance in Appeon 3.1.

- Appeon Xcelerator deployment can support large complex applications up to or even exceeding 100 megabyte in size.

- The Appeon Xcelerator deployment time is generally faster than PowerBuilder deployment to EXE. The runtime client performance of Appeon Xcelerator applications, when run in a 100Mbps internal network, almost matches the performance of PowerBuilder applications.

Note: The 100 megabyte recommendation includes the file sizes of all framework PBLs. It *excludes* the file sizes of all n-Tier NVOs used by the application, but it *includes* the file sizes of all NVOs run at the Client.

2.2 Deployment performance

2.2.1 Expected performance for full deployment

The deployment time for Appeon Pure-JavaScript applications is on par with PowerBuilder deployment to EXE. The deployment time for Appeon Xcelerator applications is 30-120% faster than Appeon Pure-JavaScript, and is as fast or faster than PowerBuilder deployment to EXE.

The table below benchmarks the performance of several applications. The applications vary in size and complexity. The deployment times are recorded for deploying the application to Client/Server with PowerBuilder and to the Web. The following notes will help you interpret the results in the table below:

Test environment was a single CPU Intel P4 2.4GHz with 1GB RAM.

- The simple application had no inheritance.
- The complex application made heavy use of inheritance (up to five levels of inheritance in some places).
- The time recorded for Appeon 3.1 is the time used for unsupported feature

analysis and Web file generation (the first two tasks in the deployment process). The performance impact of the file deployment is negligible for local deployment mode.

- The time recorded for PowerBuilder is the time to complete the entire build process and produce an EXE file with the PBDs and an EXE file with the DLLs.

Table 2-1: Appeon 3.1 Deployment Performance Benchmarks

	PB EXE w/ PBDs	PB EXE w/DLLs	Appeon Pure-JavaScript	Appeon Xcelerator
10MB simple non-PFC application	3 minutes 0 seconds	15 minutes 50 seconds	4 minutes 20 seconds	2 minutes 0 seconds
12MB complex non-PFC application	2 minutes 35 seconds	15 minutes 12 seconds	10 minutes 40 seconds	3 minutes 50 seconds
25 MB PFC application	8 minutes 5 seconds	40 minutes 6 seconds	11 minutes 50 seconds	6 minutes 50 seconds

2.2.2 Expected performance for incremental deployment

The incremental deployment functionality is used when maintaining or upgrading an already-deployed application. This feature only re-deploys to the Web the incremental changes that have been made in the application. Appeon can locate the exact lines of PowerScript code in an object that has been modified and re-deploy only those lines of code. Appeon 3.1 takes just seconds to update a single modified object.

2.2.3 Estimate full deployment time

As a general rule of thumb, with Appeon Pure-JavaScript deployment, each additional megabyte of PBLs will add 30 seconds to one minute to the full deployment process; with Appeon Xcelerator deployment, each additional megabyte of PBLs will add 10 - 30 seconds to the full deployment process. For example, it would take 15 minutes to deploy a 30MB PFC application with Pure-JavaScript deployment and 6 minutes with Appeon Xcelerator deployment. This would only apply for the initial or full deployment. All subsequent deployments should take no more than several minutes and possibly as little as a few seconds.

This rule of thumb applies to hardware configurations that meet the minimum Developer PC requirements stated in the Appeon Installation Guide. Machines with a slower clock-speed will take longer to perform the deployment.

2.3 Client runtime performance

2.3.1 Overview

The runtime performance discussed in this section refers to the performance of applications in a 100Mbps internal network that have not applied any Appeon performance boosters introduced in Section 3.2.1: *Performance boosters provided in Appeon Developer*.

Appeon Pure-JavaScript offers excellent performance when deploying small applications, good performance for medium applications, and acceptable performance, after tuning, of large and complex applications.

Appeon Xcelerator boosts the runtime performance of Appeon Web applications to levels approaching PowerBuilder Client/Server applications. This boost in runtime performance makes the Appeon Xcelerator deployment option ideal for applications that contain complex Windows with 8 or more DataWindows and heavy or complex logic on the Client, including demanding PFC applications.

2.3.2 Runtime performance with Appeon Pure-JavaScript

Appeon has tested a number of applications varying in size and complexity in Appeon Pure-JavaScript. What Appeon has found is that with some exceptions, in Appeon Pure-JavaScript Windows containing **less than 10 DataWindows*** open in **less than 5 seconds**. In contrast, Windows containing more than 10 DataWindows/DataStores take longer than 5 seconds to open. Please refer to Section 2.3.2.a: *Expected performance for non-PFC applications* and Section 2.3.2.b: *Expected performance for PFC applications* for more information.

*DataStores are non-visual DataWindows and should be counted as DataWindows.

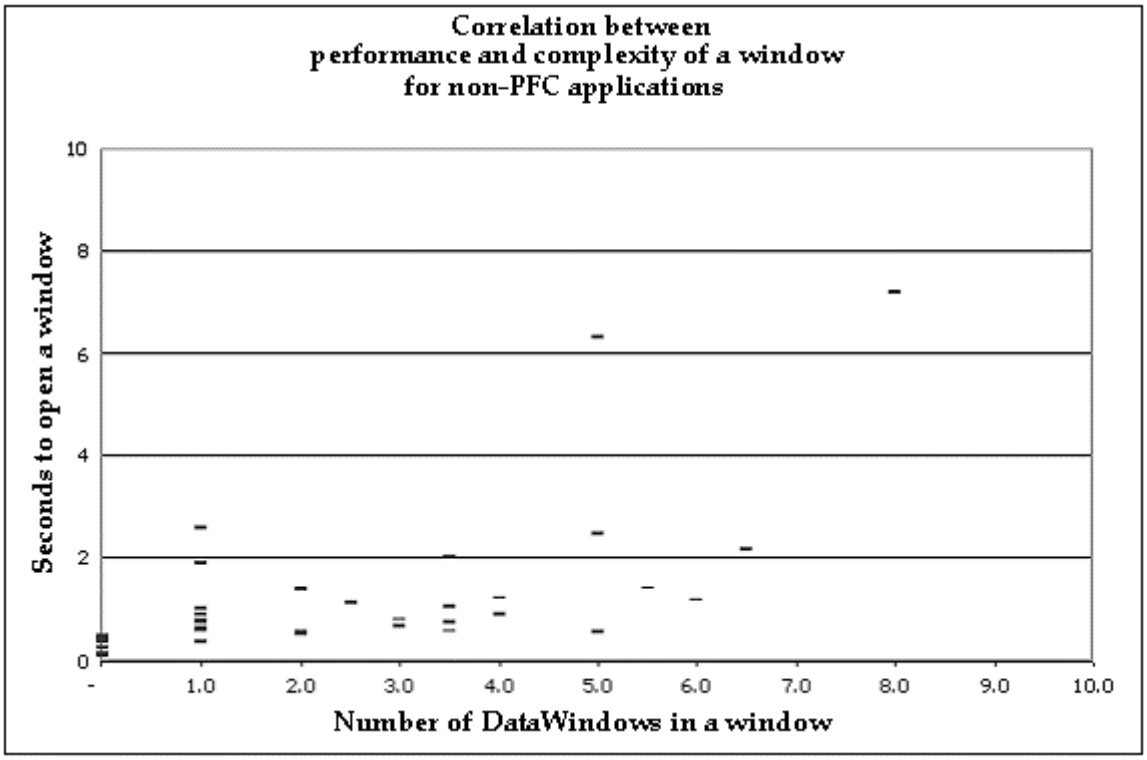
2.3.2.a Expected performance for non-PFC applications

Refer to Graph 2-1 to view the client performance of Appeon Pure-JavaScript for non-PFC applications.

The following were the test parameters:

- All applications tested were actual customer applications provided to Appeon for testing purposes. Appeon did not optimize the performance of these applications.
- Performance was measured by the number of seconds it took to open a Window and display data in the DataWindows.
- The test environment was a single CPU Intel P4 1.8 GHz machine with 512MB RAM configured for local deployment. The machine acted as the Database Server, the Application Server, Web Server, and Client. (Performance will be slower on a computer with lower clock-speed.)
- Internet download times are not considered. These results reflect a LAN environment or an Internet environment where the Web files for a given Window are cached at the Web browser. Web files are automatically cached at the Web browser after the initial Window opening.

Graph 2-1: Appeon Pure-JavaScript Runtime Performance Benchmarks for non-PFC Applications



For Appeon Pure-JavaScript, application performance in the benchmark "clusters" around 2 seconds to open a Window.

The “fewer-than-10-DataWindows/fewer-than-5-second” rule simplifies things. There are specific DataWindow features that can negatively impact performance, such as retrieving large result-sets that have many columns of data. There are also PowerBuilder features other than DataWindows that negatively impact performance, such as the number of controls/objects in the Window and the PowerScript coding style. In many situations this rule holds true for non-PFC applications, and it is the best way to get a sense of what portion of your application will not require much performance tuning.

2.3.2.b Expected performance for PFC applications

We have found that the runtime performance of Appeon Pure-JavaScript for PFC applications differs somewhat from that of non-PFC applications. The PFC framework adds some overhead, which slows down performance noticeably when compared to similar non-PFC applications. Nonetheless, many PFC Windows can still be opened in 5 seconds or less.

Refer to Graph 2-2 to view the client performance of Appeon Pure-JavaScript for PFC applications.

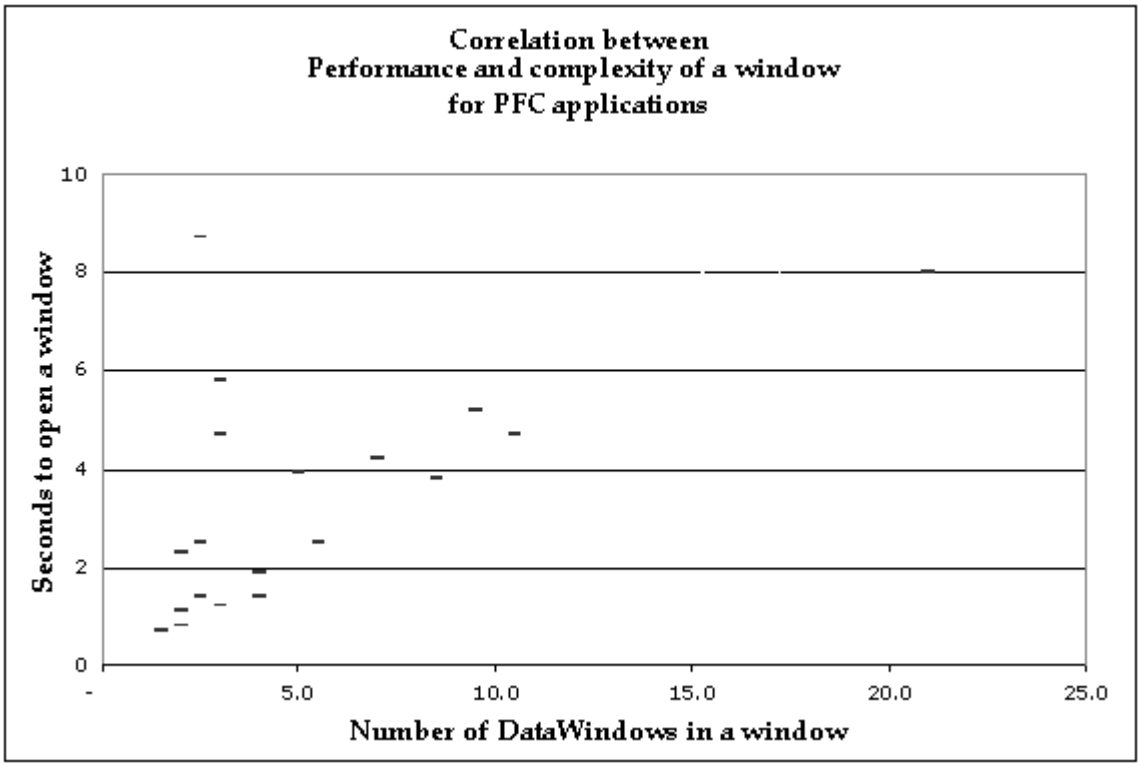
The following were the test parameters:

- All applications tested were actual customer applications provided to Appeon for testing. Appeon has not optimized the performance of these applications.
- Performance was measured by the number of seconds it took to open a Window and display data in the DataWindows.
- The test environment was a single CPU Intel P4 1.8 GHz machine with 512MB RAM

configured for local deployment. The machine acted as the Database Server, Application Server, Web Server, and Client. (Performance will be slower on a computer with a lower clock-speed.)

- Internet download times are not considered. These results reflect a LAN environment or an Internet environment where the Web files for a given Window are cached at the Web browser. Web files are automatically cached at the Web browser after the initial Window opening.

Graph 2-2: Appeon Pure-JavaScript Runtime Performance Benchmarks for PFC Applications



Based on Appeon experience working with customer PFC applications, it is possible to have many PFC Windows opened in 5 seconds or less. To achieve better performance for PFC applications, you should take the time to understand what is slowing down the Window. Then you can simplify the application by reworking or removing those performance-robbing features.

2.3.3 Runtime performance with Appeon Xcelerator

The Appeon Xcelerator deployment option boosts the runtime performance of Appeon Web applications to levels approaching PowerBuilder Client/Server. As a general rule of thumb, the runtime performance for Appeon Xcelerator applications is close to PowerBuilder Client/Server. For example, a Window that takes 1 second to open in PowerBuilder only takes 2 seconds to open on the Web with Appeon Xcelerator. This boost in runtime performance makes the Appeon Xcelerator deployment option ideal for applications with any of the following attributes:

- PFC-based applications
- Heavy Windows containing 8 or more DataWindows
- Heavy or complex logic (PowerScript) on the Client

2.3.3.a Runtime performance for non-PFC applications

For non-PFC PowerBuilder applications, the performance of Appeon Xcelerator deployment is 3-12 times faster than Appeon Pure-JavaScript deployment. The following detailed test results are for a 15MB non-PFC application that was used to run the benchmarks.

Table 2-2: Appeon Xcelerator Runtime Performance for Non-PFC Applications (in seconds)

Operation	PowerBuilder	Appeon Xcelerator	Appeon Pure-JavaScript
Launch the Web application	0.078	0.219	0.859
Administration -> Add/Delete/Modify	0.125	0.219	0.953
Administrator -> User log	0.812	0.813	2.812
File -> Available	1.656	3.453	9.093
File -> Vessel Status	1.437	1.890	6.297
Billing -> Customer	0.610	0.688	2.844
Test Environment: P4 1.6 GHz, 512MB RAM, WinXP SP2, 15MB customer non-PFC application			

2.3.3.b Runtime performance for PFC applications

For PFC-based PowerBuilder applications, the performance of the Appeon Xcelerator deployment option is 3-6 times faster than the Pure-JavaScript deployment option. The following detailed test results are for a 25MB customer PFC application that was used to run the benchmarks.

Table 2-3: Appeon Xcelerator Runtime Performance for PFC Applications (in seconds)

Operation	PowerBuilder Client/Server	Appeon Xcelerator	Appeon Pure-JavaScript
Launch the web application	0.203	2.656	6.359
Update→MIS Reserved	2.453	3.219	5.016
Report→Stock List	1.078	1.438	3.469
Test Environment: P4 1.6GHz, 512MB RAM, WinXP SP2, 24MB customer PFC application			

2.4 Performance impact of the Internet

When you move from a LAN environment to the Internet, the time it takes to open a Window increases. The opening of the Window is slowed down by the time it takes to download the files necessary for that Window to run and display data. There are two types of files downloaded. The first type is the HTML, JavaScript, and Web files that contain the UI and UI logic of the application Window. The second type is the XML files that contain the result set 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, which in some cases is slightly larger than the average 50 KB

Web page. 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. 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.

Only the XML files containing the DataWindow result sets impact performance. A result set of 50 records would result in a 12 KB XML file. Every 5 records would add another 1.2 KB to the XML file size. So, for example, a 500-record result set would correspond to a 120 KB XML file. Because these XML files contain time-sensitive data, they cannot be cached without risking the data becoming stale. Therefore, they must be downloaded from the server each time a DataWindow retrieve is initiated.

Appeon has built-in 10X data compression for DataWindow result set to essentially eliminate the time spent downloading these XML files over the Internet. The same 500-record result set that would normally correspond to a 120 KB XML file would only result in the download of a 12 KB file from the server. This compression feature makes even the largest of result sets quick to transfer over the Internet.

3 Performance Tools and Methods

3.1 Overview

There are mainly two kinds of performance tools and methods that can assist you in improving the performance of deployed applications:

- Adjustment to performance-related settings. Both Appeon and EAServer provide a number of performance-related settings that enable you to directly improve the runtime performance of Web applications without doing any performance tuning
- Performance tuning tools. Appeon provides two performance-tuning tools that help you locate the code in your application that causes performance issues. With the knowledge of the problematic code, you can do performance tuning according to the advice in Chapter 4: *Performance Tuning Basics* and Chapter 5: *Performance Fine-Tuning*.

3.2 Adjustment to performance-related settings

3.2.1 Performance boosters provided in Appeon Developer

Table 3-1 lists the performance booster options that are available in application profile configuration. To make a performance booster effective for an application, enable the option during the application profile configuration and then deploy the application with the Appeon Developer deployment wizard.

Table 3-1: Performance boosters in Appeon Developer

Performance Feature	Description	Apply To…	When To Use…
10X Web File Compression	Compresses files when they are transferred over the network.	Appeon Xcelerator Deployment	Always
Preload files into RAM on the Web Server upon startup	Enables all Web files to be loaded into RAM on the Web Server when the server is started. Each time a Client visits the application, the Web files in the server's RAM will be used, instead of the Web files on the server's hard disk.	Pure-JavaScript Deployment	In the production stage.
Allow reading files in the browser cache	Enables the Client to read Web files directly from the Internet Explorer browser cache. After the file has been downloaded the first time, the file can be cached in the browser. For subsequent requests to load the file, the Client can read it directly from the browser's cache, unless a newer version of the file is detected on the Web Server. This can greatly speed up the loading of files.	Pure-JavaScript Deployment	In debugging and production stages.

3.2.2 DataWindow Data Cache

Appeon Enterprise Manager (AEM) provides the DataWindow Data Cache tool for caching DataWindow data that are frequently used on the Web Server. To make the most of DataWindow data cache, it is recommended that you enable the cache for the DataWindow objects that do not have frequent data updates, and leave unchecked the DataWindow objects that have frequent data updates.

DataWindow Data Cache can significantly reduce server load and network traffic, boosting performance and scalability.

Important:

DataWindow Data Cache will not be effective until you fulfill all the configuration requirements that are detailed in the *DataWindow Data Cache* Section, in the *Appeon Enterprise Manager User Guide*:

- Configuration required for database servers
- Configuration required for Web servers
- Configuration for DataWindow Data Cache in AEM

3.2.3 Custom Libraries download settings

If your application uses a customer library, you can specify how the custom library should be downloaded to the client in the AEM Custom Libraries tool:

- If the file size of the custom libraries is large, you should set the install mode to "Install manually (no automatic installation)". With this option, Appeon does not handle the DLL and OCX files installation for the application. Users must manually install the DLL and OCX files of the application before accessing it. Therefore, the time for downloading the DLL and OCX files from the Web server to the client can be saved.
- If the file size of the custom libraries is small, you can set the install mode to "Install automatically without asking user" or "Confirm with user, then install automatically". With these options, the DLL and OCX files will be downloaded from the Web server to the client when the application is run.

For more details about the custom libraries download settings, refer to the *Modifying Custom Libraries Install Settings* Section in the *Appeon Enterprise Manager User Guide*.

3.2.4 File caching in Internet Explorer

For optimal performance, it is recommended that the Web file caching functionality of Internet Explorer be fully used. 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 get the best performance while safeguarding your application from becoming "stale" (such that out of date cache settings override recent changes).

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 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.

Now the browser is set to automatically check for newer versions of a Web application. The browser will also check to see that there is enough allocated space in the Temporary Internet Files folder to allow for caching Appeon Web applications as well as other Web applications and Web sites that you may use.

Note: If you experience problems or errors when accessing a Web application after deployment, the cached Web application files may be the cause. There are two ways to solve this problem:

- Check the *Empty Temporary Internet Files folder when browser is closed* option (under the Advanced tab of Internet Options) to ensure that no cached files remain whenever Internet Explorer is restarted, or
- Manually delete the temporary Internet files: go to Tools | Internet Options, click the *Delete Files…* button, restart Internet Explorer, and try again.

Allowing the temporary Internet files folder to be emptied each time the browser is closed is an option that is recommended for use only during the development stage. During development, it is best to have the latest Web application loaded in the browser to avoid any problems caused by cached files. When your Web application is ready for production deployment, this setting can be disabled (unchecked). This setting should be reset once development is complete so that the Web application can be cached at each Client PC for better scalability and Client-side performance. Caching files should be enabled in a production environment and disabled in a development/testing environment.

3.2.5 Performance of Appeon Server

Because Appeon Server is installed to EAServer, a basic rule is: the better the EAServer's performance is tuned, the better Appeon Server's performance. You can refer to the *EAServer Performance and Tuning Guide* for details on how to do performance settings in EAServer. This section highlights several key performance settings.

3.2.5.a EAServer JVM startup option

When starting EAServer, the -jvmtype switch can specify that the client, server, or classic Java VM be used. It is recommended that you set the switch to the Java server VM, because it is tuned for application server performance.

3.2.5.b Configuring connection caches

JDBC driver used by EAServer connection cache

Unless your application database is Sybase ASA or ASE, you should avoid using the JDBC-ODBC driver as the driver for connection caches. Instead, use the Native-protocol/all-Java driver. For details, refer to *JDBC driver preparation* Section in the *Appeon Migration Guide*.

Reportedly, the Microsoft JDBC driver is quite slow. If your database is Microsoft SQL 2000, you may consider using a SQL Server JDBC driver other than MS JDBC driver, or you can add sendStringParametersAsUnicode=false to JDBC URL to increase performance.

Tuning the cache size

Connection caches have 10 connections by default. For applications with many clients, this number is often too small. For lightly used caches, you can lower the size to free up memory

and network connections that would be wasted by rarely used database connections; for heavily used caches, you can higher the size.

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. If the maximum number is set improperly it will degrade performance.

Refer to the *EAServer Performance and Tuning Guide* for details on tuning the cache size.

3.2.5.c EAServer performance tuning: how to set the http properties

You should configure the EAServer http.maxthreads and server.maxconnections properties such that it can handle the expected concurrent 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 optimum results. That is why it is essential that stress testing simulate the expected number of concurrent users.

3.2.5.d Turning off Appeon Server log

After the application debugging is done, turn off the Appeon Server log functionality in AEM for better Appeon Server performance.

3.2.6 Database performance tuning

There are various techniques for fine-tuning your database server performance. You can refer to the documentation provided with database servers for more details on database performance tuning. This section highlights several key database performance settings.

3.2.6.a Database indexes

Effective indexes are one of the best ways to improve performance in a database application. A table scan happens when there is no index available to help a query, and on large tables, scans have a significant impact on performance.

Determining the correct indexes to use in a database requires careful analysis, benchmarking, and testing. There are a number of guidelines to building the most effective indexes for your application.

- Short Keys: Having a short index is beneficial for two reasons. First, database

work is inherently disk intensive. Second, larger index keys will cause the database to perform more disk reads, which limits throughput.

- Distinct Keys: The most effective indexes are those with a small percentage of duplicated values.
- Covering Queries: An index contains all of the columns needed for a query so that the database can save a disk read without returning to the table for more information.
- Clustered Indexes: A clustered index is closer in similarity to a phone book because each index entry contains all the information you need

3.2.6.b Adjusting database parameters

You need to carefully adjust database parameters (for example, the database cache size) according to the instructions of the database server performance-tuning document.

3.3 Performance tuning tools

3.3.1 Heavy Window report

The Heavy Window Report is a function in Appeon Developer to help identify "Heavy" Windows. Heavy Windows are Windows in the PowerBuilder application that most affect the performance of the deployed Web application. These Windows tend to be stuffed with data-intensive PowerBuilder controls/objects that negatively impact performance such as DataWindows and TreeViews. As a result, they may take significantly longer than 0.5 second to open.

By using the heavy Window analysis report, the user can get a general idea of how many heavy Windows are contained in the application. This report lists the oversize JavaScript files generated for the current application and the heavy windows, including the number of complex controls and the minimum load value of each window. The report will even specify the composition of the Window, such as the number and types of controls it contains. With this information you will know exactly which Windows to tackle first and what PowerBuilder features to thin out. The report is fully configurable so that you can specify how "heavy" a Window can be and what PowerBuilder features make the Window "heavy". Refer to the *Appeon Developer User Guide* for more information.

3.3.2 Web performance runtime tracing report

The runtime tracing report gives you a second way to organize your efforts in improving Web performance. You need to know the most time-consuming "hot spots" in your application. The "hot spot" may not include a heavy Window, but it may include a less frequent cause of slow performance such as interlinked objects and controls or complex business rules.

Perform the following steps to use this report.

STEP 1 – Run the Web application.

STEP 2 – Click the *Information* button () on the Appeon Developer toolbar.

STEP 3 – Click *Performance* in the left treeview and the *Web Performance Report* button will be displayed on the right of the page.

STEP 4 – Click the *Web Performance Report* button to open the report.

The Web Performance Runtime Tracing Report Window is displayed, as shown in Figure 3-1. The Web Performance Runtime Tracing Report window automatically refreshes every 15 seconds, dynamically displaying the latest runtime performance of a running Web application.

Figure 3-1: Web Performance Runtime Tracing Report

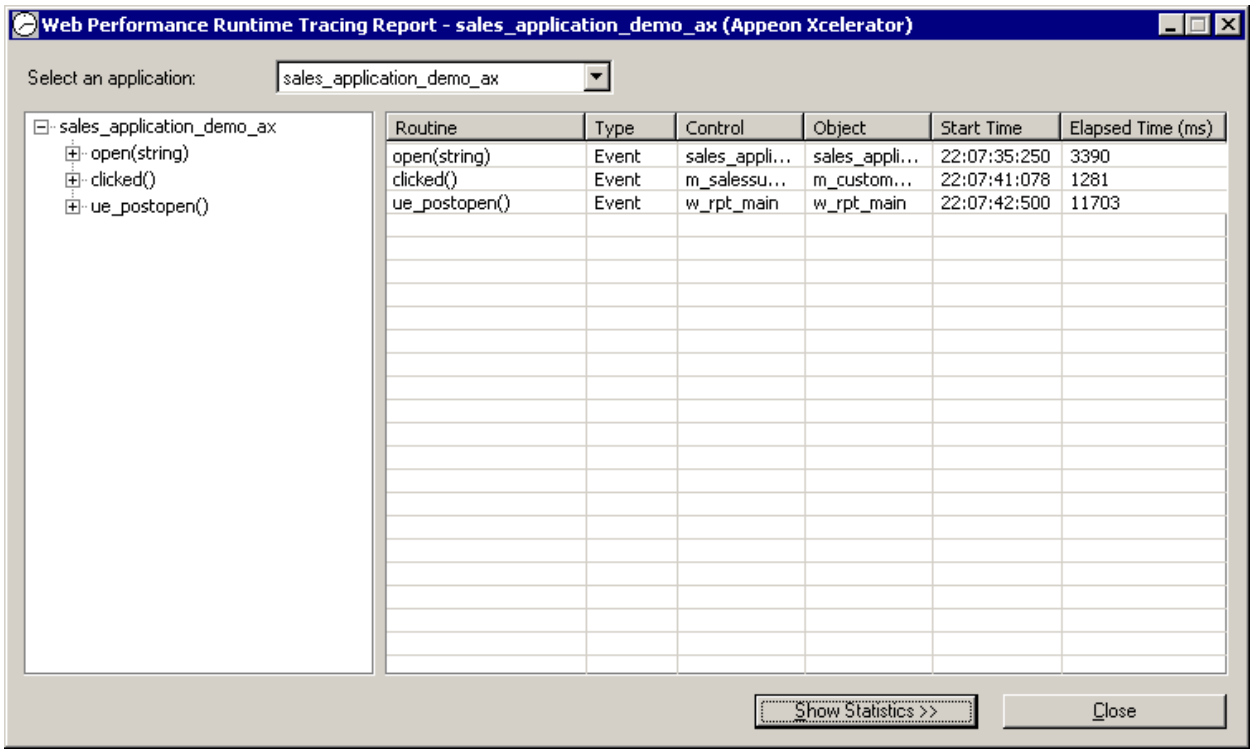
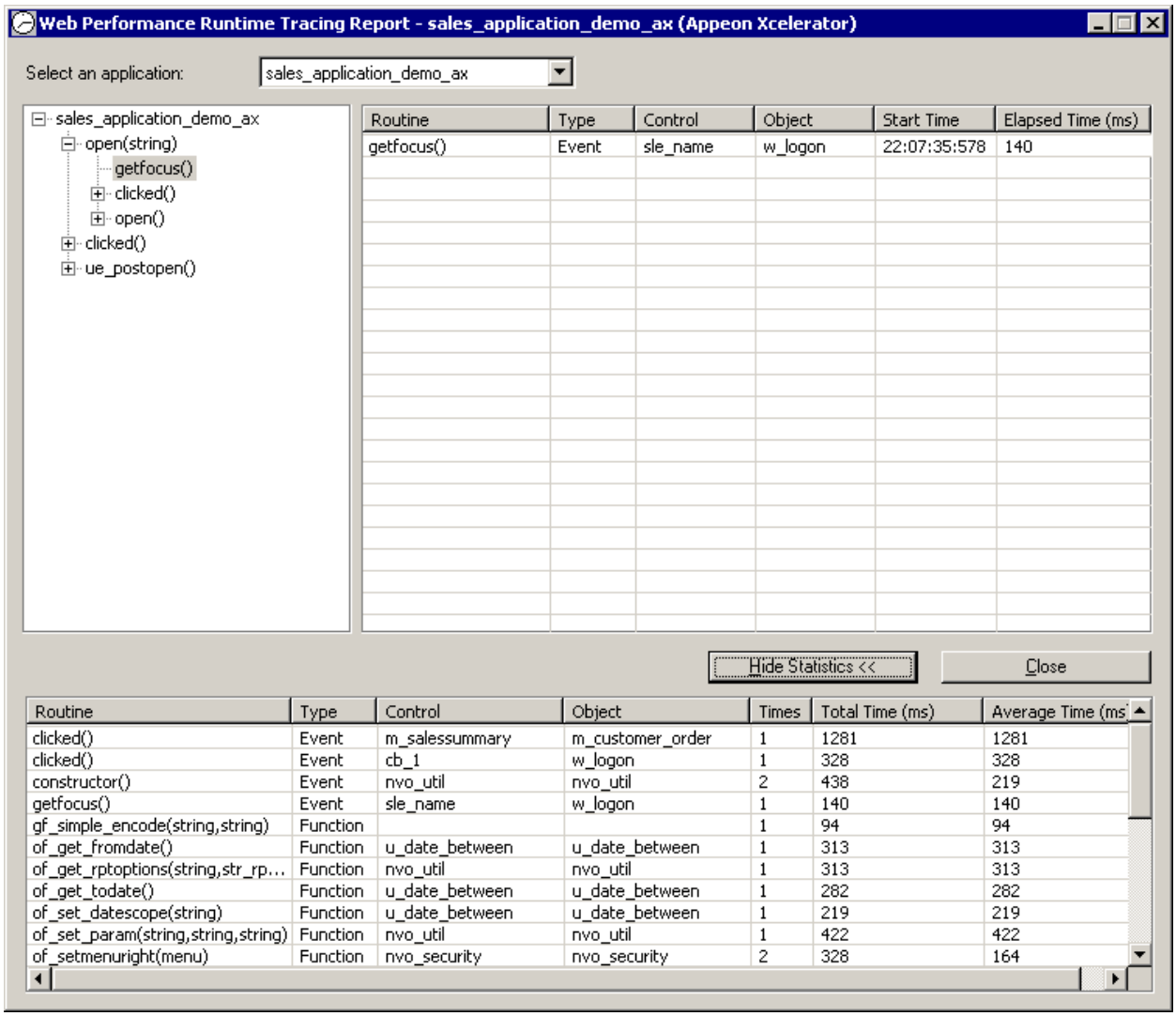


Table 3-2: Web Performance Runtime Tracing Report

Option	Description
Application list box	Lists the application profiles configured in Application Profiles. The default application profile is selected.
Triggered event treeview	Displays the triggered events in a treeview structure. You can expand and select the items in the treeview to view detailed information of the executed events/functions in the right box.
Routine	Displays the name of the event/function that is triggered.
Type	Displays the type of the triggered routine, event or function.
Control	Displays the control that contains the routine.
Object	Displays the root object that contains the routine.
Start Time	Displays the time at which the routine is called.
Elapsed Time	Displays the elapsed time used to complete the execution of a routine.
Show/Hide Statistic	Click this to show or hide the summary of each routine: the number of calls, the total elapsed time, and average time used in each call.

For example, click the getfocus() event under the open(string) in the tree view. Detailed information is displayed in the right box as shown in Figure 3-2. From the Elapsed Time column, you can find out which routines impede performance.

Figure 3-2: Web Performance Runtime Tracing Report



Expand the report by clicking the *Show Statistics* button at the bottom of the window. The statistics shown in the expanded box summarize the time each routine is called, and the total elapsed time and average time used in each call, as shown in Figure 3-2.

4 Performance Tuning Basics

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 can be done?

Table 4-1 shows the three most common causes of poor Web-performing applications ranked from the most common to the least. These three causes affect both Appeon Xcelerator performance and Pure-JavaScript deployment performance.

Table 4-1: The three most common causes of poor Web-performing applications

Priority	Overall Cause	PB Features Involved	What To Do	See Section
1	Excessive data retrieval	DataWindows, DropDownDataWindows and non-visual database access using SQL (Structured Query Language)	Break up retrieval of large result sets into small subsets that can be displayed on a single screen; also, consider obtaining fewer columns	4.1.1
2	Heavy Windows	DataWindows, DropDownDataWindows and TabPages	Reduce size by factoring (breaking up) Windows into more manageable units	4.1.2
2	Overuse of global functions and variables	Windows, NVOs and PowerScript global variables	Find out where the variables or objects are used: if in one place, make the variable or object local; if in multiple places consider passing the variable or object as a parameter or using a storage area that belongs exclusively to the user routines.	4.1.3
3	Intermingled and complex business logic	PowerScript	Place repeated business logic in one consolidated NVO and use this object where the business logic is needed, or record the evaluation result in one place (ideal for performance)	4.1.4

4.1 Common causes of poor-performing applications

4.1.1 Excessive data retrieval

The single most common issue Appeon has found with PowerBuilder applications that perform poorly is the retrieval into DataWindows and non-visual objects of large result sets consisting of many rows, often with many columns.

Correspondingly, reduction of the size of data sets requested in these applications, especially

their size in rows but also their size in columns, makes dramatic performance improvements!

Section 5.1: *Reducing the size of data set requests* of this document discusses this issue in detail.

4.1.2 "Heavy" Windows

Another common issue Appeon has found with PowerBuilder applications that perform poorly is that they contain "heavy" Windows. Two factors determine the "heaviness" of a Window:

- The number of objects and controls (especially DataWindows, DropDownDataWindows, and TabPages) in the Window.

  Appeon has included Tabs in this list because Tabs encourage a large number of DataWindows and other data-intensive controls to be stuffed into a single Window.

- The complexity of the controls in the Window. For example, a DataWindow with several DropDownDataWindows makes the Window significantly heavier than a DataWindow without DropDownDataWindows.

  A DataWindow will be more complex and have greater impact on the heaviness of the Window if it contains more computed columns, more property expressions, and more EditMask, DropDownDataWindow, or DropDownListBox columns.

"Heavy" Windows will open slowly on the Web. Generally, they will take five seconds or more. The runtime performance of the Window may also be affected if the controls/objects are chained or related.

4.1.3 Large "global" overhead

The second most common issue is the Internet Explorer memory consumption or overhead caused by too many "global" PowerBuilder features such as global functions and global variables, or the overhead caused by large complex frameworks. "Global" features by definition are made available throughout the application. Since they must remain resident in memory, they consume Internet Explorer memory. For example, each additional global function or global variable will consume Internet Explorer memory even if it is not being actively used. As more Internet Explorer memory is consumed, the Web application runs more slowly.

Note: There are two reasons to consider reducing global overhead. Not only do unnecessary global functions and variables consume memory and cause the Web application to run slowly, they also are a common source of application failure. This is because methods in the application can accidentally or deliberately make inappropriate references to global variables and global objects.

These inappropriate references reduce performance, and they make the behavior of the method dependent on the availability and the status of the global variable. This means that other changes will potentially interact with the reference to make the final application unpredictable and unreliable.

The ability of any method to access and alter the unnecessary global overhead increases the probability of application failure in the statistical sense because it widens the number of distinct states the application may enter. Only a small percentage of those states will be acceptable. For this reason it is often a good idea to analyze global variables and objects to make certain that they need to be global.

The simplest case occurs when a global variable is used once and nowhere else. In this case it should be local to the procedure that uses it. If it is a variable, it should be placed in the PowerScript subroutine or function that uses the variable.

When the global variable occurs in more than one place, determine whether they have a common storage area that is not global and use this area instead. Ideally, this area should only be visible to the routines that access its objects and variables.

Alternatively, consider removing the variable from global storage and making it a parameter of the routines that need it.

This analysis can pay off in efficiency and reliability.

4.1.4 Intermingled and complex business logic

The third most common issue is intermingled and complex business logic. Many PowerBuilder applications have a single business rule coded in several places, such as in Windows, DataWindows, and controls. Appeon has seen applications where the execution of a single business rule fires off more than 20 events!

Intermingled and complex business logic at the Client side negatively impacts performance in several ways. First, complex logic run at the Client side takes longer to process than complex logic run at the server side (in n-Tier NVOs with C++-based PBVM rather than JavaScript). Secondly, logic on the Client side consumes Internet Explorer memory while logic on the server does not. As more Internet Explorer memory is consumed, the Web application runs more slowly. Finally, intermingled business logic takes longer to execute than logic that is consolidated at one location contained in an NVO.

Intermingled and complex business logic also makes for an application that is harder to maintain and understand. This is because the repetition of the business rules may not be completely precise.

In one place, for example, the actual PowerScript rule may be If income<20000 Then approveMicroLoan. In another, the rule may be If income<=20000 Then approveMicroFinancing. Thus, there may be two methods (with different logic in detail), approveMicroLoan and approveMicroFinancing with the same intent. When two of our customers compare notes, however, one finds that they have been approved while the other finds that they have been turned down when their situations are the same. The application is seen as unreliable and unsafe.

While this type of business logic is always harder to maintain and keep correct, intermingled and complex business logic significantly impacts performance only if it is used extensively at the Client side. Many PowerBuilder applications do not have a lot of business logic – only DataWindows and some UI logic. Other applications have implemented the business logic layer mostly as database stored procedures. Finally, the nature of some applications do not contain much business logic other than the basic UI and validation logic such as data entry applications. Even though these types of applications may contain some intermingled business logic, this should not be very problematic given the small amount and the simplicity of the logic at the Client.

Consider reviewing intermingled, complex, and repeated business logic in cases where it frequently appears for this reason, since it can, as in the above case of global variable analysis, simultaneously improve both the performance and reliability of your applications.

It may not be necessary to constantly re-evaluate the same business logic over and over if you can evaluate the rule once and store its result in a variable. You can also consider placing the

repeated logic so that it is called where needed. This will not address the original performance issue but it will address the issue of maintainability and reliability. In fact, in this particular case, the calling mechanism will decrease performance by a slight margin.

4.2 Tuning the features that commonly degrade performance

4.2.1 Thin-out "heavy" Windows

Redesign the navigation strategy to present a lighter-weight Client user interface.

- Reduce the number of DataWindows. In many situations, the DataWindows in a Window can be spread out across multiple Windows, thereby reducing the "weight" of the Window. Appeon suggests that a Window contain less than five DataWindows. DataStores are non-visual DataWindow objects so each DataStore in the Window should also be counted as a DataWindow.
- Reduce the number of DropDownDataWindows. Read-only DropDownDataWindows that are used for selecting data in a field of a DataWindow can be re-implemented using Embedded SQL. Two DropDownDataWindows are equivalent to one DataWindow in terms of their negative impact on runtime performance. Therefore, each DropDownDataWindow is equal to 0.5 of a DataWindow in weight.
- Use individual Windows instead of Tabs. The Tab control encourages a large number of DataWindows to be stuffed into a single Window. Appeon recommends reworking each Tab page into its own individual Window that is accessed through a menu. As a general rule of thumb, using Tab control in a Window that contains more than two DataWindows in addition to other controls is not recommended.
- Reduce complex TreeViews. TreeViews that contain a lot of data and have deep hierarchy may perform well. However, they will take up a lot of Internet Explorer memory at the Client which will negatively affect the overall performance of the application. It is recommended that the number of items in a TreeView not exceed 200.

4.2.2 Decrease "global" overhead

Reduce Internet Explorer memory consumption by decreasing the "global" overhead.

You can decrease the "global" overhead by reducing the number of "global" PowerBuilder features such as global functions and variables. For example, in many situations it may be possible to replace global functions with object functions, and replace global variables with local or instance variables. It will further improve performance if the object functions are in NVOs that are deployed to Appeon Server. These can be instantiated in the Window as needed or be run from the server.

As has been noted above, this provides a twofold benefit. The application performs better and is more reliable and easier to maintain because it does not contain global variables and objects which increase the probability of application failure and are more difficult to track.

You can also decrease "global" overhead by decreasing the size of the application framework. If there are services in the framework not used by the application, you can boost performance by deleting those unused NVOs/services from the framework PBLs. If an NVO/service is not used extensively throughout the application, it may improve performance if the service is

moved out of the framework and only implemented locally where it is needed. Of course, this is not the most ideal workaround since it will result in the duplication of the service if the service is used in more than one place; you need to be sure that the use of the service is precisely the same wherever it is used.

4.2.3 Consolidate business logic into NVOs

Consolidate business logic into NVOs and deploy these NVOs to Appeon Server. This partitioning of business logic will significantly thin out the Client-side processing. The Client-side will be faster and Web files will be lighter.

For example, Appeon worked with a Japanese conglomerate whose application was taking more than 10 seconds to execute a DataWindow update. The customer had coded significant amounts of validation rules, each validation rule having very complex logic. This was not the simple "make sure the field is not empty" sort of validation. Appeon helped this customer consolidate these business rules into n-Tier NVOs. As a result, the 10+ second DataWindow update was slashed to one second. As you can see, n-Tier NVOs are very powerful and Appeon strongly encourages you to use them if your application is large and complex.

4.3 What to do next

Performance tuning requires a good understanding of the application code. This chapter has laid out the basics of performance tuning, which is the quickest and easiest way to significantly boost performance. The next chapter, Performance fine-tuning, drills down into various PowerBuilder features and coding styles known to degrade performance. This will require you to dive further into the existing application code, but it will allow you to boost performance. It is especially recommended that you look carefully at the next chapter if you have Windows that contain fewer than five DataWindows but take significantly longer than five seconds to open when deployed to the Web.

5 Performance Fine-Tuning

This section describes many strategies you can apply to rapidly bring your Web-enabled PowerBuilder application up to speed. Below, Table 5-1 identifies the most important (Priority 1), less important (Priority 2), less critical (Priority 3) and least critical (Priority 4) changes, which are then described in detail in this section.

Table 5-1 is your checklist for performance fine-tuning. However, always keep in mind that a "least critical" change or a "less important" change can have a dramatic and positive effect in a high use procedure or object in your application. See Section 3.3.2: *Web performance runtime tracing report* for the use of the Runtime Tracing Report which helps you isolate high use areas in your application.

Unless specified explicitly, the performance fine-tuning suggestions are applicable for both Appeon Xcelerator deployment and Pure-JavaScript deployment. However, it is not often necessary to fine-tune Appeon Xcelerator deployments due to the performance advantage in Appeon Xcelerator (refer to Section 2.3.3: *Runtime performance with Appeon Xcelerator*).

Table 5-1: Fine-tuning strategy checklist in priority order

Priority	Change	Overall Guideline	See Section
1	Reduce the size of data set requests	Get small amounts of data at a time	5.1
1	Reduce client-server interactions	Minimize the number of calls to the server.	5.8
2	Global definitions	Reduce global definitions	5.3.7
2	Tabs in Heavy Windows	Don't use Tabs in Heavy Windows	5.6.1
2	OpenSheet	Where possible use OpenSheet instead of Open	5.6.2
2	DataWindow columns	Reduce columns in DataWindows	5.7.4
3	Nesting level of an object	Reduce both static and dynamic nesting to four	5.2.3
3	Inheritance	Keep the inheritance level under four	5.2.2
3	Initialization	Minimize code in Open, Resize, Active, and Constructor events	5.4.2
3	Autoinstantiation	Whenever possible use non auto-instantiated controls	5.5.5
3	DataWindows	Reduce the number of DataWindows in Heavy Windows	5.6.1
3	DropDownDataWindows	Reduce the number of DropDownDataWindows in Heavy Windows	5.6.1
3	Retrieving data into DataWindows	Reduce rows in DataWindows; use to_date function for retrieve date values from Oracle	5.7.1

3	Data synchronization	Use ShareData or RowsCopy and RowsMove	5.7.2
3	Data sorting	SQL's ORDER BY clause preferable to the DataWindow Sort property	5.7.3
3	Computed fields	Minimize these fields which trigger re-computation in many events	5.7.5
3	Complex filtering	Filters in a DataWindow with one or more function calls trigger events unexpectedly	5.7.8
3	DataStore	Use DataStore and not DataWindow for non-visual processing	5.7.10
3	DataWindow SQLPreview event	Reduce usage of the SQLPreview event in a DataWindow	5.7.12
3	Composite DataWindows	Improve the performance of Composite DataWindows	5.7.14
4	Minimize Loops	Minimize looping, especially loops that refresh data or the presentation at the client	5.3.1
4	Arrays	Minimize complicated and special arrays	5.3.2
4	Review calculations	Copy function results into variables before executing loops	5.3.5
4	Dynamic calls	Avoid dynamic calls	5.4.1
4	Events	Avoid using events that are triggered repeatedly	5.4.3
4	Batch Ops	Use batch operations	5.4.4
4	Submenus	Limit submenu levels Note: Submenu levels are not limited in Appeon Xcelerator deployment.	5.5.3
4	TreeView and ListView	Efficiency tips for TreeView and ListView	5.5.4
4	Slow DataWindow edit styles	DropDownDataWindows and EditMask	5.7.6
4	DataWindow expressions	Minimize DataWindow expressions	5.7.7
4	Describe and Modify are "good"	Use Describe and Modify to get and set DataWindow object properties	5.7.9
4	RowsFocusChanging/RowsFocus Changed	Minimize code in these two events	5.7.13
5	Any is "bad"	Avoid and remove Any type variables	5.3.3
5	TreeView inside Heavy Windows	Reduce usage of TreeView inside Heavy Windows	5.6.1
5	Dot notation	Avoid and remove dot notation, and use simple variables and extra code	5.3.4
5	Shared names	Avoid sharing names among variables	5.3.6
5	Advanced controls	Reduce number of advanced controls	5.5.1

5	Picture-based controls	Reduce and optimize picture-based controls	5.5.2
5	Cross/inter-Window calls	Avoid cross-Window and inter-Window calling	5.6.3
5	SetRow	Use SetRow instead of ScrollToRow	5.7.11

5.1 Reducing the size of data set requests

5.1.1 Size of requests for data

Appeon has found that the single factor most affecting performance when applications move to the Web is the size of requests for data, typically made either with SQL in the application program, or in stored procedures.

Of course, in the Client-Server and mainframe world, it often made sense in database applications to get all the data needed by the application in one request. The database operation was often “atomic”, and locked against interference by competing requests as well as against the modification of the data that was being retrieved, thereby assuring the data’s integrity.

But take a look at any popular and well-visited Web site such as Google. When you use Google, or Yahoo for that matter, to search for information, the presentation of the first page contains up to about 10 results no matter how many “real” results exist, and, the response, especially on Google (a leader in search technology) is *fast*.

This may be because the search engine at Google requests only the first 10 results.

Of course, we're discussing a proprietary technology and we can only speculate. Nonetheless experience confirms that you need to consider getting data "just in time".

The user may make, in your company’s database, a request for your company’s equivalent of a popular Google search such as “pet store”, “vacation”, or “Appeon solutions”. But just as the average Google is probably not interested in pet stores located in Mukden, unless she lives there, your corporate user will want records ranked in such a way that performance is “just in time” for the most useful results.

Follow the “just in time” philosophy in regard to your own Web-enabled applications. Consider adding SQL WHERE clauses based on more search criteria, and retrieving only the amount of data that will fit on a typical screen. You can expose a Next button, and have the application respond to this button click by getting the next logical segment. Refer to the Section 5.7.1: *Speeding up data-retrieving process*, for more instructions on how to display data in a DataWindow in steps.

5.1.2 Retrieval of an excessive number of columns

Another frequent source of lowered performance is the retrieval of an excessive number of columns, whether explicitly, or implicitly.

A sign of the implicit retrieval of excess 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 the explicit retrieval of excess 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.

While requesting excess columns does not have the large performance impact as a request for excess rows, it does tend to slow down nested loops in your application which process rows in the outer loop, and columns in the inner loop. This is because the overall time of the outer row-wise loop is multiplied, for each row, by the number of columns. Reducing the number of columns requested to the precise set of columns needed can correspondingly reduce your Web-enabled application timings.

5.2 Simplifying application complexity

Complexity of the application is an important factor that determines whether an application will perform well when deployed to the Web. You will be able to boost runtime performance by reducing the number of complex objects in the application and in a given Window.

5.2.1 Object reference

There are two types of object references in PowerBuilder applications.

- Object reference type 1: The object makes references in the script. For example, in the script of object A, there are three references: to object B, object C, and object D. In this case, the number of cross-references by object A is 3.

  Note: This object reference type has an impact on runtime performance on Appeon Pure-JavaScript deployment but no impact on performance on Appeon Xcelerator deployment. In Appeon Pure-JavaScript deployment, a Window, Menu, or UserObject object is considered as complex if the number of object references it makes is more than ten.

- Object reference type 2: A window or visual user object contains complex visual object(s) such as DataWindow controls, TreeView controls, ListView controls, Tab controls, or Custom visual objects.

  A Window or visual user object containing complex visual object(s) such as DataWindow or TreeView controls, is complex if the number of object references exceeds four (object reference type 2).

5.2.2 Inheritance

Keep the inheritance level under four. If an object's inheritance level is over four, Appeon considers it a complex object that will result in degraded performance. In addition, it is highly recommended that you avoid using inheritance for basic controls such as StaticText, CommandButton, SingleLineEdit, and MultiLineEdit.

To save memory that the application will occupy, it is recommended that the base class object only perform generalized processing. Do not attempt to place ALL the logics in the base class object of a hierarchy of ancestor objects and descendent objects when each of the descendent objects only uses a small portion of the logics.

5.2.3 Nesting level

There are two types of nesting in PowerBuilder applications:

- Nesting type 1: When object A refers to object B, and object B refers to object C. In this case, the nesting level of object A is three.

  An object is considered as a complex object if the nesting level of the object is more than five.

- Nesting type 2: When object A contains object B, object B contains object C, and the script accesses to a property or function of object C via object A. The nesting level of the script is three.

  For example, a Window *w_main* has a Menu object *m_mymenu*, and *m_mymenu* has a File menu with an Open item. You can access the Visible property of the Open item with script *w_main.m_mymenu.m_file.m_open.Visible*. In this case, the nesting level of the script is four.

  An object is considered as complex if more than 1/3 of the event/function calls in the object have 4+ nesting levels of type 2.

5.3 PowerScript tuning

5.3.1 Minimizing loops

Excessive loops have a negative impact on performance. Some functions cause the Appeon Web application to redraw/refresh Web DataWindow/controls. If such functions are put into a loop, it will slow the speed of painting the Web UI.

Appeon suggests that you minimize or eliminate excessive loops using the following methods:

- In the Open event of a Window, avoid using a loop for retrieving data into DataWindows.
- 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)
...
```

5.3.2 Minimizing complicated and special arrays

Complicated arrays such as Structure arrays, User Object arrays, and Nested arrays are supported, as are special arrays such as Constant array. However, these arrays negatively impact performance. Appeon suggests that you minimize the usage of such arrays.

5.3.3 Using a defined standard data type instead of any data type

Use of Any data type slows performance because Appeon must perform additional processing to determine the actual data type. In particular, the Web performance is noticeably affected if an Any variable is used in a loop. Appeon recommends that you use a defined standard data type instead of the Any data type.

Sometimes it is necessary to use Any to represent so-called polymorphic data. For example, in an order entry application, the customer may want to specify either an absolute number of units required, or, in a "just in time" business situation, where some units are always better than none. In this example, the customer may desire to enter "at least 10 units but not more than 20".

Ideally, and especially in the client-tier, this would be represented by an object that is able to represent both absolute quantities and ranges.

However, actual enterprise case studies show that rapid application development pressures cause client-side developers to represent these polymorphic values with Any variable types. In the case of our order entry example we can discover, while upgrading a legacy

Client/Server application to the Web, that the developers, pressed for time, represent the order quantity as an absolute number (when the customer orders a precise number), or as two numbers separated by a space (when the customer orders a minimum and a maximum range).

Performance is negatively affected as more work needs to be done to obtain the value. Reliability is also affected because unlike an object, the Any variable can in fact be in any type.

Therefore, replacing Any variables with if-then logic that chooses the appropriate representation, or making them into non-visual objects, can enhance the performance and reliability of your application.

5.3.4 Reworking Dot Notation expressions with variables

Because of the overhead in processing dot notation as seen in objectName.methodName, large amounts of dot notation expressions can impact performance. Using variables is 10 to 20 times faster.

When you repeatedly use the same dot notation expression, you can improve performance by defining a variable and assigning a partial reference to the variable. This is especially useful in a loop, and it can eliminate hundreds or thousands of repetitive executions of a dot notation expression.

The following is an example in which the row number is obtained and used to obtain several column numbers.

```
ls_column1 = dw_1.GetItemString(dw_1.GetRow(),"column1")
 ls_column2 = dw_1. GetItemString(dw_1.GetRow(),"column2")
      ……
 ls_column10 = dw_1. GetItemString(dw_1.GetRow(),"column2")
```

```
Long ll_row
ll_row = dw_1.GetRow()
ls_column1 = dw_1.GetItemString(ll_row,"column1")
ls_column2 = dw_1.GetItemString(ll_row,"column2")
      ……
ls_column10 = dw_1.GetItemString(ll_row,"column10")
```

Note that moving the row number into a 32-bit long variable increases performance. The application no longer needs to (1) access the object, or (2) perform the GetRow property or method code several times.

This can increase performance greatly in cases where the properties and methods accessed do repetitious error checking that only needs to be performed once.

5.3.5 Reworking calculations with variables

It is much faster to use variables than to call functions to perform a calculation. If the result of a calculation or a function is used often, assign the result to a variable and use the variable where the calculation or the function is called. This can improve efficiency of calling the calculation by up to 50%.

The following is an example:

Original code:	`For I = 1 To dw_1.RowCount()`

Optimized code:	`ll_rowcnt = dw_1.RowCount()` `For I = 1 To ll_rowcnt`

Note also that this change can increase the reliability of the For loop as translated to JavaScript. This is because in PowerScript the *ll_rowcnt* function will be evaluated once at the start of the For loop while in the corresponding JavaScript code, the function will be evaluated each time at the beginning of the loop.

In the unlikely, but possible, event that the row count changes, the two versions of the code, PowerBuilder versus JavaScript, will give different results in the original code situation. They will provide identical results in the optimized code situation.

5.3.6 Avoiding sharing names among variables

If one name is shared among two or more variables, the Web application must perform additional processing to locate the proper variable. Appeon recommends that you avoid sharing names among different types of variables.

5.3.7 Reducing global definitions

Global variables and global functions remain resident in memory. This consumes more Internet Explorer memory which can significantly slow down performance. Where possible, Appeon suggests replacing global variables with local or instance variables, and replacing global functions with object functions. It will further improve performance if the object functions are in NVOs that are deployed to Appeon Server. These can be instantiated in the Window as needed, or be run from the server.

5.4 Events and functions

5.4.1 Avoiding dynamic calls

Do not use dynamic calls to functions or events. Instead, replace these dynamic calls with static calls. This will improve performance somewhat, especially if several dynamic calls are made to complete a process.

5.4.2 Minimizing code in Open, Resize, Active, and Constructor events

Performing large amounts of initialization work, especially when opening a Window, will significantly degrade performance. Appeon recommends that you minimize the code in the following events:

- Open: specifically for opening Windows. If possible, move a portion of initialization work into a user-defined event that is triggered after the Window has opened.
- Resize: it is strongly recommended that you not resize a Tab control.
- Activate
- Constructor

5.4.3 Avoiding using events that are triggered 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). Appeon recommends that you minimize the code or the usage of events with high repetition such as Timer, MouseMove, SelectionChanging, GetFocus, LoseFocus, Activate, and Deactivate.

5.4.4 Using batch operations

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!)
```

5.5 Controls and objects

5.5.1 Reducing number of advanced controls

Reducing the number of the following advanced controls will improve performance: Nested UserObject, Nested Tab, Tab, TreeView, ListView, EditMask, and Line.

5.5.2 Reducing and optimize picture-based controls

Picture-based controls consume additional system resources and bandwidth. Therefore, do not use picture controls unless necessary. Always set the picture format to GIF unless the picture is a photograph or the image has 256 or more colors, in which case, for best results, use the JPEG format. Also, avoid swapping pictures for the Picture, PictureButton, PictureListBox, or ListView controls.

5.5.3 Limiting submenu levels

Appeon Pure-JavaScript applications have a limit on the number of submenu levels to ensure good performance, but Appeon Xcelerator applications do not have this limit. Each additional level of submenus negatively impacts performance. In general, a few levels of submenus will not noticeably impact performance. However, menus with more than five submenu levels will have a negative impact on performance.

5.5.4 Using TreeView and ListView more efficiently

TreeView and ListView are complicated controls that consume significant system resources and tend to delay the painting of the Web UI. If you need to use TreeView and ListView, Appeon recommends the following:

1) If there are several levels and items in a TreeView control, do not initialize the entire TreeView in the Window "open" event. Instead, initialize the top level first, and once the Window has been fully displayed, gradually initialize the lower levels.

2) Frequently adding/removing the TreeView and ListView entries significantly impacts performance. Appeon suggests that you limit this operation on a TreeView or ListView to no

more than 10 times for a given event or user action.

5.5.5 Using non-autoinstantiated objects whenever possible

Use non-autoinstantiated objects rather than autoinstantiated objects whenever possible. Non-autoinstantiated objects have much better performance then autoinstantiated objects. For example, using a non-autoinstantiated object to pass an argument is 20 times faster than using an autoinstantiated object.

5.6 Window

5.6.1 Thinning out "heavy" Windows

Redesign navigation strategy to present a lighter-weight Client user interface.

Reduce the number of DataWindows. In many situations, the DataWindows in a Window can be spread out across multiple Windows, thereby reducing the "weight" of the Window. Appeon suggests that a Window contain fewer than five DataWindows. DataStores are non-visual DataWindow objects, so each DataStore in the Window should also be counted as a DataWindow.

Reduce the number of DropDownDataWindows. Read-only DropDownDataWindows that are used for selecting data in a field of a DataWindow can be re-implemented using Embedded SQL. Two DropDownDataWindows are equivalent to one DataWindow in terms of their negative impact on runtime performance; therefore each DropDownDataWindow should be counted as 0.5 DataWindow.

Use individual Windows instead of Tabs. The Tab control adds 1-2 seconds to the opening of the Window. Furthermore, the Tab control encourages a large number of DataWindows to be stuffed into a single Window. Appeon recommends reworking each Tab page into its own individual Window that is accessed through a menu. As a general rule of thumb, using a Tab control in a Window that contains more than two DataWindows plus other controls is not recommended.

Reduce usage of TreeViews. A TreeView control adds 1-2 seconds to the opening of the Window. This may be tolerable if the Window is not already close to being "heavy" (fewer than four DataWindows and no Tab control). However, TreeViews that contain a lot of data and deep hierarchy will not perform very well. If TreeViews are used, it is recommended that the TreeView be kept small.

5.6.2 Using OpenSheet instead of Open

Compared to the Open function, the OpenSheet function is twice as fast in opening a Window. Use the OpenSheet function to open Windows whenever possible.

5.6.3 Avoiding cross-Window calling or inter-Window calling

Cross-Window and inter-Window calls negatively impact performance. Instead of cross-Window calls and inter-Window calls, use parameters for passing the required information to the calling Window.

The following is an example:

```
OpenSheet(w_2, w_mdi, 0, original!)
//In the clicked event of button b_1 in the Window w_1
```

```
ls_value = w_1.is_value   //In the open event of the Window w_2
```

```
OpenSheetWithParm(w_2, is_value)
//In the clicked event of button b_1 in the Window w_1
```

```
Ls value = message.StringParm   //In the open event of the Window
w_2
```

5.7 DataWindow

5.7.1 Speeding up data-retrieving process

Runtime performance suffers in applications that have large quantities of data (more than 20 columns and 1000 rows) in DataWindows. The reasons are: (1) Browser/Server architecture is totally different from Client/Server architecture in the handling of data; (2) Data transfer over the network, especially the Internet, can be quite slow.

If the PowerBuilder application contains a large amount of data, it is recommended that you make the following changes before converting the application with Appeon:

- Reduce the amount of data by removing data that is unnecessarily used in the Web application.
- Add a more detailed SQL WHERE clause for retrieving data.
- Instead of retrieving all the data into a DataWindow at one time, retrieve data into the DataWindow in steps. Section 5.7.1.a: *Retrieving data into a DataWindow in steps* gives you instructions on how to achieve this.
- 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 Section 5.7.4: *Reducing the number of columns in DataWindows*) can, in many cases, improve performance to an even greater extent than a larger reduction in rows. This is because most of the time, loops, whether in the application code or in the PowerBuilder VM, visit columns first and then rows.
- Separate the data operation from other operations. For example, do not retrieve a large amount of data into a DataWindow directly during the DataWindow painting process.

5.7.1.a Retrieving data into a DataWindow in steps

For Oracle database:

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.

For databases other than Oracle:

If your application uses a different 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.

5.7.1.b Using to_date function for retrieving date values from Oracle

It is recommended not to directly use a Date parameter in a SQL statement for retrieve data from Oracle database. Appeon internal logic calls to the to_timestamp function for the Date parameter before retrieving, which slows down performance.

You can call the Oracle to_date function in the SQL statement to avoid directly using the Date parameter.

The following is an example:

```
WHERE ("T_PRO_PRO_PROLIST"."F_PLAY_DATE" = :F_PLAY_DATE)
```

```
WHERE ("T_PRO_PRO_PROLIST"."F_PLAY_DATE" =
To_Date(:F_PLAY_DATE,"yyyy-mm-dd"))
```

Note: This performance-tuning suggestion only applies to Appeon 3.1. The above-described Appeon internal logic will be modified in the next version.

5.7.2 Using ShareData or RowsCopy/RowsMove for data synchronization

The following PowerBuilder functions can synchronize data between two DataWindows: ShareData, RowsCopy/RowsMove, Object.Data, and SetItem. The execution performance of the functions on the Web, from the fastest to the slowest, follows this sequence:

ShareData > RowsCopy/RowsMove > Object.Data > SetItem

5.7.3 Using "Order by" instead of Sort property for data sorting

Avoid using the DataWindow Sort property; instead, use "Order by" in the SQL statement. Their functions are the same, but it is proven that the execution of an SQL statement is faster than the execution of a sort function or the Sort property assignment. For example, there is a test DataWindow with one column and 20,000 records, and the DataWindow sorts by the column. With the DataWindow Sort property set, it takes more than 10 seconds to retrieve

into the Web DataWindow. After removing the Sort property and adding an "Order by" clause to the DataWindow SQL statement, the retrieval time is reduced to 3 seconds.

5.7.4 Reducing the number of columns in DataWindows

It takes time to initialize columns and controls in a DataWindow. Moreover, the more columns there are in a DataWindow, the slower it is to retrieve and display data. Therefore, Appeon recommends the following:

- Remove unnecessary columns and controls from DataWindow.

  If you have a choice between reducing the number of columns retrieved, and reducing the number of rows, note that a small reduction in columns can, in many cases, improve performance to an even greater extent than a larger reduction in rows (described above in Section 5.7.1: *Speeding up data-retrieving process*). This is because most of the time, loops, whether in the application code or in the PowerBuilder VM, visit columns first and then rows.

- If the Visible property of a column is set to zero (the control is not visible), and the column is not used at all, it is best to remove it to reduce the DataWindow size.

5.7.5 Avoiding 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*".

5.7.6 Reworking DropDownDataWindows and reduce EditMask edit styles

There are two DataWindow edit styles that are significantly slower than others: DropDownDataWindows and EditMasks. For example, DropDownDataWindows are generally 40% slower than DropDownListBoxes to display data. Appeon suggests that you avoid using EditMasks and DropDownDataWindows. Use other DataWindow edit styles such as RadioButtons, Edit, and DropDownListBoxes.

Some DropDownDataWindows can be worked around. If the DropDownDataWindow is only for displaying data (without requiring the user to select or input a value), you can replace it with the Edit style and use SQL statements to achieve the same functionality (from the end-user perspective). Best of all, you will not be degrading performance.

Appeon suggests that the number of records in a DropDownDataWindow should not exceed 1,000.

5.7.7 Reducing usage of DataWindow expressions

Generally speaking, when performing a DataWindow Retrieve function, the execution time

of the DataWindow without property expressions is at least twice as fast. Therefore, Appeon recommends that you reduce usage of DataWindow expressions, 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.

5.7.8 Avoiding complex filtering

Complex filtering means that the filter criteria contain one or more expressions that call to function(s). It is recommended that you not use complex filtering on a DataWindow, especially on a DataWindow that has large amounts of data (more than 20 columns, more than 1000 rows).

Avoid using date/datetime columns in filter, sort, and find functions.

5.7.9 Using Describe and Modify to get and set DataWindow object properties

Dot notation is less efficient than the Describe and Modify functions. In general, the Describe and Modify functions are two to three times faster than dot notation. Therefore, Appeon recommends that you use the Describe function to replace dot notation that gets the DataWindow object properties, and that you use the Modify function to replace dot notation that sets the DataWindow object properties. It is also recommended that you not use the Modify function too often.

5.7.10 DataStore is preferred over DataWindow for non-visual data processing

A DataStore is a non-visual DataWindow control. When you perform operations on DataStore, the Web application does not need to do any UI work, so it is faster than a DataWindow. DataStore is over two times faster than DataWindow in performing data retrieval. Appeon recommends that you use DataStores instead of DataWindows for non-visual data processing whenever possible.

5.7.11 Using SetRow instead of ScrollToRow

The DataWindow SetRow and ScrollToRow functions perform similar functionalities; they both change the current row in DataWindow. ScrollToRow scrolls a DataWindow to the specified row, which is UI-related. This consumes additional time. Appeon recommends that you use SetRow instead of ScrollToRow to locate the specified row.

The following is an example:

```
dw_employee.ScrollToRow(10)
```

```
dw_employee.SetRow(10)
```

5.7.12 Reducing usage of DataWindow SQLPreview event

Each time the DataWindow SQLPreview event is triggered, the Web application will interact with Appeon Server twice. This takes 1-2 seconds. Appeon recommends that you minimize

writing script into the SQLPreview event of the DataWindow.

5.7.13 Minimizing code in RowsFocusChanging and 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. Appeon recommends that you do not write code into RowsFocusChanging and RowsFocusChanged events unless it is necessary.

5.7.14 Performance tuning for Composite DataWindows

Composite DataWindows usually take longer to open and operate in PowerBuilder and on the Web. The following suggestions are mainly targeted for performance tuning applications deployed with Appeon Xcelerator, but can also be valid for applications deployed in Pure-JavaScript:

- For composite and nested reports, the number of calls to the server increases with the number of retrieval arguments specified for the child report(s) and DropDownDataWindow(s). Avoid using retrieval arguments for child reports and DropDownDataWindows in Composite and Nested reports.
- Avoid setting the Height.Autosize property to enabled.
- Avoid dynamically changing the Height.Autosize, Position.Y, Trail_Footer, and Report properties.
- Reduce the nesting level of Composite DataWindows. The opening and repainting performance of a Composite DataWindows decreases proportionally with the nesting level. If the first nesting level contains 100 rows, and the second nesting level contains 100 rows, the performance of the DataWindow will be even slower than a DataWindow containing 10,000 rows.
- Avoid creating a DataWindow with a data row greater than the size of the target page.
- Avoid using DeleteRow and InsertRow functions. The following example shows how an InsertRow function affects the performance of a Composite DataWindow. If it takes Y time to open a Composite DataWindow that contains one row, when executing the following code for the DataWindow, the execution time will be around Y * (1 + 2 + 3 +…+ 100) = 5500*Y

```
For l  = 1 to 100
      InsertRow(0)
      …
Next
```

5.8 Reducing client-server interaction

Each call from the client to the server takes one second. You can improve performance significantly by minimizing the number of calls to the server.

You can minimize the number of calls to the server by limiting the usage of these statements:

- Retrieve and Update functions for DataWindow and DataStore.

In Crosstab, N-Up, and Label DataWindows, page scrolling usually involves multiple calls to the server.

- Select statements
- Cursor statements
- Call for stored procedures

You should especially avoid using these statements in a loop.

Another way to reduce the number of Client-Server interactions is to make full use of n-Tier NVOs. Encapsulate all business logic, dynamic SQL statements, and database update statements into n-Tier NVOs. In this way, the previous dozens of Client-Server interactions can be reduced to one, and the server side execution can be tight and efficient.

The following types of scripts can be encapsulated in n-Tier NVOs for efficient server side execution:

- Database operations.
- Retrieve, Update, and especially retrieving or updating multiple DataWindows at the same time.
- Events and functions.

  The events and functions that contain complex dynamic SQL statements, complex embedded SQL statements, Cursor statement, or call Stored Procedures.
- Business logic.
- High-level validation of updated data.
- Synchronization of data update.
- A series of data computations caused by an update.
- Transaction management.
- Complex computation.
- Numeric computation that is complex or has high-precision requirements.
- Non-numeric computation such as sort, search, and data selection.
- Appeon unsupported features (for example, dynamic SQL format 4).

Index

1

10X data compression, 10

A

about this book, 1
accessing Web applications problem, workaround
  checking the Empty Temporary Internet Files folder when browser is closed option, 13
  deleting the temporary Internet files manually, 13
Active, Resize, Open and Constructor events, minimizing code, 30
adjustment to performance-related settings, 11
advanced controls, reducing, 31
Appeon Server, performance, 13
Appeon Xcelerator deployment, 4
application complexity, simplifying
  inheritance, 26
  nesting level, 27
  object reference, 26
arrays
  minimizing complicated and special arrays, 28
audience, 1

B

business logic, consolidating into NVOs, 22
business logic, intermingled and complex, poor performing, 20

C

calculations, reworking with variables, 29
changes for Appeon 3.0, client runtime, 5
client runtime performance
  changes for Appeon 3.0, 5
client runtime performance, 5
client-server interaction, reducing, performance tuning, 38
columns in DataWindow, reducing, 35
complex filtering, avoiding, 36
Composite DataWindows, performance tuning, 37
computed fields, avoiding, 35
Constructor, Active, Resize and Open events, minimizing code, 30
controls and objects, performance tuning
  advanced controls, reducing, 31
  picture-based controls, reducing and optimizing, 31
  submenu levels, limiting,, 31
  TreeView and ListView, using efficiently, 31
  using non-autoinstantiated objects, 32
cross-window calling and inter-window calling, avoiding, 32
Custom Libraries download settings, 12

D

Database performance
  using indexes, tuning, 14
DataStore or DataWindow, performance tuning, 36
DataWindow
  performance tuning
    Order by, sorting data, 35
DataWindow Data Cache, 12
DataWindow expressions, reducing,, 36
DataWindow SQLPreview event, reducing, performance tuning, 37
DataWindow, performance tuning
  columns, reducing, 35
  complex filtering, avoiding, 36
  Composite DataWindows, 37
  computed fields, avoiding, 35
  DataWindow expressions, reducing, 36
  DropDownDataWindows, EditMask edit styles, reworking and reducing, 35
  retrieving data into a DataWindow in steps, 33
  rows, reducing, 33
  RowsFocusChanging and RowsFocusChanged events, minimizing code, 37
  SetRow or ScrollToRow, 36
  SQLPreview event, reducing, 37
  using DataStore, 36
  using Describe and Modify, 36

DataWindow, performance tuning
ShareData or RowsCopy/RowsMove, synchronizing data, 34
deployment options
Appeon Xcelerator deployment, 4
Pure-JavaScript deployment, 4
deployment options, 4
deployment performance, 4
deployment performance
estimating full deployment times, 5
full deployment
expected performance, 4
incremental deployment
expected performance, 5
Dot Notation expressions, reworking with variables, 29
DropDownDataWindows and EditMask edit styles, performance tuning, 35
dynamic calls, avoiding, 30

E

EAServer JVM startup option, settings, EAServer performance, 13
EAServer performance, configuring
EAServer JVM startup option, settings, 13
http properties, settings, 14
JDBC driver used by EAServer connection cache, settings, 13
events and functions, performance tuning
dynamic calls, avoiding, 30
events that are triggered repeatedly, avoiding, 31
Open, Resize, Active and Constructor events, minimizing code, 30
using batch operations, 31
excessive data retrieval, poor performing, 18

F

file caching in Internet Explorer, 12
fine-tuning , performance, 23
full deployment
estimating time, 5
expected performance, 4

G

global definitions, reducing, 30
global overhead, decreasing, 21
global overhead, poor performing, 19

H

heavy window report, 15
heavy windows, poor performing, 19
heavy windows, thin-out
DataWindows, reducing, 21
DropDownDataWindows, reducing, 21
individual windows and Tabs, 21
TreeViews, reducing, 21
how to use this book, 1
http properties, settings, EAServer performance, 14

I

if you need help, 3
incremental deployment
expected performance, 5
inheritance, simplifying application, 26
Internet Explorer configuration
caching recommendation
disk space to use, 13
temporary Internet files folder, Advanced tab, 12
temporary Internet files, General tab, 12
file caching recommendation, 12

J

JDBC driver used by EAServer connection cache, settings, EAServer performance, 13

L

ListView and TreeView, using efficiently, 31
loops, minimizing, 27

N

nesting level, simplifying application, 27
non-PFC applications
expected performance, 6
non-PFC applications runtime performance, 9

O

object reference, simplifying application, 26

Open, Resize, Active and Constructor
events, minimizing code, 30
OpenSheet or Open, performance tuning, 32

P

performance fine-tuning, 23
performance impact of Internet, 9
performance impact of Internet
10X data compression, 9
performance levels, 4
performance of Appeon Server, 13
performance of Database, 14
performance of EAServer, 13
performance tools and methods, 11
performance tuning
controls and objects
advanced controls, reducing, 31
picture-based controls, reducing and optimizing, 31
submenu levels, limiting, 31
TreeView and ListView, using efficiently, 31
DataWindow
columns in DataWindow, reducing, 35
complex filtering, avoiding, 36
Composite DataWindows, 37
computed fields, avoiding, 35
DataWindow expressions, reducing, 36
DropDownDataWindows, EditMask edit styles, reworking and reducing, 35
rows in DataWindow, reducing, 33
RowsFocusChanging and RowsFocusChanged events, minimizing code, 37
SetRow or ScrollToRow, 36
ShareData or RowsCopy/RowsMove, synchronizing data, 34
SQLPreview event, reducing, 37
using DataStore, 36
using Describe and Modify, 36
using Order-by, sorting data, 35
events and functions
dynamic calls, avoiding, 30
events that are triggered repeatedly, avoiding, 31
Open, Resize, Active and Constructor events, minimizing code, 30
using batch operations, 31
PowerScript tuning
avoiding sharing names among variables, 30
calculations, reworking with variables, 29
defined standard data type or any data type, 28
Dot Notation expressions, reworking with variables, 29
global definitions, reducing, 30
loops, minimizing, 27
minimizing complicated and special arrays, 28
reducing the retrieval of an excessive number of columns, 25
reducing the size of requests for data, 25
server settings and performance
client-server interaction, reducing, 38
simplifying application complexity
inheritance, 26
nesting level, 27
object reference, 26
using non-autoinstantiated objects, 32
what to do next, 22
window
cross-window calling and inter-window calling, avoiding, 32
heavy windows, thin-out, 32
OpenSheet or Open, 32
performance tuning basics, 18
performance tuning tool
heavy window report, 15
Web performance runtime tracing report, 15
performance tuning tools, 15
performance-boosters
10X Web File Compression, 11
allow reading files in the browser cache, 11
preload files into RAM on the Web Server, 11
performance-boosters, 11
performance-related settings, adjustment, 11
PFC applications
expected performance, 7
PFC applications runtime performance, 9
picture-based controls, reducing and

optimizing, 31
poor-performing applications, causes
business logic, intermingled and complex, 20
excessive data retrieval, 18
heavy windows, 19
large global overhead, 19
poor-performing applications, tuning
business logic, consolidating into NVOs, 22
global overhead, decreasing, 21
heavy windows, thin-out, 21
PowerScript tuning
avoiding sharing names among variables, 30
calculations, reworking with variables, 29
defined standard data type or any data type, 28
Dot Notation expressions, reworking with variables, 29
global definitions, reducing, 30
loops, minimizing, 27
minimizing complicated and special arrays, 28
Pure-JavaScript deployment, 4

R

readers, 1
Resize, Open, Active and Constructor events, minimizing code, 30
retrieving data into a DataWindow in steps, 33
rows in DataWindow, reducing, 33
RowsFocusChanging and RowsFocusChanged events, minimizing code, 37
runtime performance
Appeon Xcelerator, 8
client, 5
Pure-JavaScript, 6
runtime performance with Appeon Xcelerator
non-PFC applications runtime performance, 9
PFC applications runtime performance, 9
runtime performance with Pure-JavaScript
non-PFC applications expected performance, 6
PFC applications expected performance, 7

S

server settings and performance
client-server interaction, reducing, 38
SetRow or ScrollToRow, performance tuning, 36
Sort property, avoiding, 35

T

thin-out heavy windows. See heavy windows, thin-out
tools and methods, performance, 11
TreeView and ListView, using efficiently, 31
tuning basics, performance, 18

U

using batch operations, performance tuning, 31
using Describe and Modify, setting DataWindow object properties, 36
using non-autoinstantiated objects, performance tuning, 32

V

variables names, avoiding sharing, 30

W

Web performance runtime tracing report, 15
window, performance tuning
cross-window calling and inter-window calling, avoiding, 32
heavy windows, thin-out, 32
OpenSheet or Open, 32