Microsoft BizTalk Server 2010 Performance Optimization Guide

Microsoft Corporation

Published: February, 2011

Summary

The BizTalk Server 2010 Performance Optimization Guide contains prescriptive guidance for optimizing BizTalk Server performance, based upon hands-on experience of IT professionals who have worked extensively with BizTalk Server. This guide contains four main sections: Finding and Eliminating Bottlenecks, Optimizing Performance, Scaling a Production BizTalk Server Environment, and BizTalk Server Performance Testing Methodology.

Microsoft

Copyright

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Contents

Microsoft BizTalk Server 2010 Performance Optimization Guide

Welcome to the Microsoft® BizTalk® Server 2010 Performance Optimization Guide. We created this guide to provide in-depth information for optimizing the performance of a BizTalk Server solution. Full end-to-end performance testing is frequently overlooked during enterprise application deployment. Knowing that Microsoft has built a scalable messaging infrastructure, many organizations that use BizTalk Server spend little or no time conducting performance testing of their own applications. BizTalk Server applications consist of many parts, which may include custom-built components as well as those provided by Microsoft. It is impossible for Microsoft to performance test every possible combination of these components. Therefore, fully and properly conducting a performance test of your application is a critical step of any deployment.

The purpose of this guide is to consolidate and provide prescriptive guidance on the best practices and techniques that should be followed to optimize BizTalk Server performance.

To download a copy of this guide in chm, pdf, or docx form, go to http://go.microsoft.com/fwlink/?LinkID=209185.

What's in it?

Generally, the performance of a server is determined by the component that has the lowest performance—the bottleneck in the system. The key to improving performance is being able to identify bottlenecks, determine their cause, and apply the appropriate corrective action.

As you plan your BizTalk Server 2010 deployment, use this guide to help design and optimize your environment. The concept of performance is closely related to the concept of scalability. When you have a solid understanding of the factors influencing the performance of system components, you can deploy components in a way that scales to support periods of high demand.

This guide provides guidance for optimizing performance, based upon hands-on experience of IT professionals who have worked extensively with BizTalk Server. Specifically, this guide includes four main sections:

- **Finding and Eliminating Bottlenecks**: The Finding and Eliminating Bottlenecks section describes various types of performance bottlenecks as they relate to BizTalk Server solutions and information about how to resolve the bottlenecks.
- **Optimizing Performance**: The Optimizing Performance section provides guidance for optimizing performance of a BizTalk Server solution. BizTalk Server performance is closely tied to performance of the platform upon which BizTalk Server is installed. This section provides recommendations for optimizing performance of both BizTalk Server and the BizTalk Server platform.
- **Scaling a Production BizTalk Server Environment**: The Scaling a Production BizTalk Server Environment section provides detailed results of BizTalk Server 2010 performance

testing completed by the BizTalk product team. These tests focused on scalability and measured the impact of adding BizTalk Server 2010 computers, the impact of adding BizTalk Server MessageBox databases, and the impact of adding both BizTalk Server 2010 computers and BizTalk Server 2010 MessageBox databases to a solution simultaneously.

  - When increasing the number of BizTalk Server computers in a BizTalk Server group, for these tests only one BizTalk Server MessageBox database was configured for the BizTalk Server group. These tests focused solely on the impact of adding BizTalk Server computers to a BizTalk Server group.
  - When increasing the number of BizTalk Server MessageBox databases used by the BizTalk Server group. These tests focused solely on the impact of adding BizTalk Server MessageBox databases to a BizTalk Server group.
  - When increasing the number of both BizTalk Server computers and BizTalk Server MessageBox databases used by the BizTalk Server group. These tests measured the impact of adding both adding BizTalk Server computers and BizTalk Server MessageBox databases to a BizTalk Server group.

- **BizTalk Server Performance Testing Methodology**: The BizTalk Server Performance Testing Methodology section provides detailed information about how to test and optimize BizTalk Server performance. It includes information about which performance criteria to focus on and the fundamental phases that should be applied when assessing BizTalk Server performance.

Additions to this version of the guide

Using Visual Studio to Facilitate Automated Testing – Describes the use of Visual Studio Load testing to evaluate the performance of a BizTalk Server application.

Acknowledgments

We in the BizTalk Server User Education team gratefully acknowledge the outstanding contributions of the following individuals for providing both technical feedback as well as a good deal of content for the BizTalk Server 2010 Performance Optimization Guide:

Authors

- Tim Wieman, Microsoft
- Paolo Salvatori, Microsoft
- Trace Young, Microsoft

Reviewers

- Tim Wieman, Microsoft
- Paolo Salvatori, Microsoft

Finding and Eliminating Bottlenecks

A successful BizTalk Server performance assessment is largely a matter of discovering the existence of, and then resolving, bottlenecks to accommodate either more throughput or reduced

latency. This section describes various types of performance bottlenecks as they relate to BizTalk Server solutions and information about how to resolve the bottlenecks.

In This Section

- Best Practices for Avoiding Bottlenecks
- Investigating Bottlenecks
- System-Level Bottlenecks
- Bottlenecks in the BizTalk Server Tier
- Bottlenecks in the Database Tier
- Performance Tools

Best Practices for Avoiding Bottlenecks

While the default settings in BizTalk Server provide optimal performance for many hardware and software configurations, in some scenarios it may be beneficial to modify the settings or deployment configuration. When configuring BizTalk Server, consider the following performance guidelines:

- To prevent resource contention, isolate receiving, orchestration, and sending on separate hosts. To further minimize contention, isolate the tracking service from other hosts.
- If CPU processing on the computer running BizTalk Server is the bottleneck, scale up the computer running BizTalk Server by including additional CPUs or upgrading to faster CPUs.

SQL Server Guidelines

Consider the following performance guidelines when configuring Microsoft SQL Server with BizTalk Server:

- Whenever possible, use a fast disk subsystem with SQL Server. Use a redundant array of independent disks type 10 (RAID10/0+1) or a storage area network (SAN) with backup power supply.
- Isolate each MessageBox database on a separate server from the BizTalk Tracking database (BizTalkDTADb). For smaller deployments if CPU resources are available, it might be sufficient to isolate the MessageBox database on a separate physical disk from the BizTalk Tracking database.
- The primary MessageBox database could be the bottleneck due to CPU processor saturation or latency from disk operations (average disk queue length). If CPU processing is the bottleneck, add CPU processors to the primary MessageBox. If not, try to disable publishing on the master MessageBox database. This way the master MessageBox database can more efficiently handle routing of messages to the other MessageBox databases. The option to disable publishing is valid when you are using multiple MessageBox databases.
- If disk operations are the bottleneck, move the BizTalk Tracking database to a dedicated SQL Server computer and/or dedicated disk. If CPU processing and disk operations on the primary MessageBox database are not the bottleneck, you can create new MessageBox databases on the same SQL Server computer to leverage your existing hardware.

- Follow recommendations in Optimizing Filegroups for the Databases to isolate the transaction and data log files for the MessageBox and BizTalk Tracking databases onto separate physical disks.
- Allocate sufficient storage space for the data and log files. Otherwise SQL Server will automatically consume all of the available space on the disks where the log files are kept. The initial size of the log files depends on the specific requirements in your scenario. Estimate the average file size in your deployment based on testing results, and expand the storage space before implementing your solution.
- Allocate sufficient storage space for high-disk-use databases, such as the MessageBox, Health and Activity Tracking (HAT), and Business Activity Monitoring (BAM). If your solution uses the BizTalk Framework messaging protocol, allocate sufficient storage space for the BizTalk Configuration database (BizTalkMgmtDb).
- Depending on business needs, such as data retention periods, and the volume of data processed in your scenario, configure the "DTA Archive and Purge" SQL Server Agent job on the HAT-Tracking database such that the BizTalk Tracking database does not grow too large. The growth of this database can degrade performance because reaching the full capacity of the database imposes a limit on the rate of data insertion. This is especially true when one BizTalk Tracking database supports multiple MessageBox databases.
- Scale up the servers hosting the MessageBox and BizTalk Tracking databases if they are the bottleneck. You can scale up the hardware by adding CPUs, adding memory, upgrading to faster CPUs, and using high-speed dedicated disks.
- Splitting the TempDB files across multiple files may resolve performance issues related to I/O operations. As a general guideline, create one file data file per processor and use the same size for all files created.
- Change the database auto-grow settings to a fixed value such as 100-150MB. By default the database growth is configured to 10%, which can lead to delays when growing larger databases.
- SQL Server memory should be set to a fixed value by setting both Min Server Memory and Max Server Memory to the same value. In general, allocate 75% of physical memory to SQL Server and leave 25% for the rest of the operating system and any applications. If this is a dedicated SQL Server, you can decrease the amount reserved for the operating system to a minimum of 1GB.

See Also

Finding and Eliminating Bottlenecks

Investigating Bottlenecks

This topic describes a recommended process for investigating bottlenecks.

What is the source of the problem?

The source of the bottleneck could be hardware or software related. When resources are underused, it is usually an indication of a bottleneck. Bottlenecks can be caused by hardware limitations, by inefficient software configurations, or by both.

Identifying bottlenecks is an incremental process whereby alleviating one bottleneck can lead to the discovery of the next one. The science of identifying and alleviating these bottlenecks is the objective of this topic. It is possible for a system to perform at peaks for short periods of time. However, for sustainable throughput a system can only process as fast as its slowest performing component.

Using an iterative approach to testing

Bottlenecks can occur at the endpoints (entry/exit) of the system or in the middle (orchestration/database). After the bottleneck has been isolated, use a structured approach to identify the source. After the bottleneck is eased, it is important to measure performance again to ensure that a new bottleneck has not been introduced elsewhere in the system.

The process of identifying and fixing bottlenecks should be done in an iterative manner. Vary only one parameter at a time, repeat exactly the same steps during each test run, and then measure performance to verify the impact of the single modification. Varying more than one parameter at a time could conceal the effect of the change.

For example, changing parameter 1 could improve performance. However, changing parameter 2 in conjunction with changing parameter 1 could have a detrimental effect and negate the benefits of changing parameter 1. This leads to a net zero effect and results in a false negative on the effect of varying parameter 1 and a false positive on the effect of varying parameter 2.

Testing consistency

Measuring performance characteristics after changing settings should be done to validate the effect of the change.

- **Hardware -** Use consistent hardware because varying the hardware can cause inconsistent behavior and produce misleading results. For example, you would not use a laptop to test performance of a BizTalk solution.
- **Test Run Duration -** Measure performance for a fixed minimum period to ensure that the results are sustainable. Running tests for longer periods also ensures the system has gone through the initial warm/ramp up period where all caches are populated, database tables have reached expected counts, and throttling is given sufficient time to regulate throughput once predefined thresholds are hit. This approach will help discover optimal sustainable throughput.
- **Test Parameters –** Do not vary test parameters from test run to test run. For example, varying map complexity and/or document sizes can produce different throughput and latency results.
- **Clean State -** After a test is complete, ensure that the state of the test environment is clean before running the next test. For example, historical data can build up in the database affecting run-time throughput. Recycling the service instances helps to release cached

resources like memory, database connections, and threads. In your test environment, you may want to create and execute the bts_CleanupMsgbox stored procedure as described in How to Manually Purge Data from the MessageBox Database in a Test Environment (http://go.microsoft.com/fwlink/?LinkId=158064). This script is intended to return your BizTalk Server test environment to a fresh state with regards to the Message Box between runs. The script deletes all running instances and all information about those instances including state, messages, and subscriptions, but leaves all activation subscriptions so you do not have to re-enlist your orchestrations or send ports. Note that this tool is not supported on production systems.

- **Performance Testing and Tuning -** The goal of this test category is to maximize performance and throughput of your application and find the Maximum Sustainable Throughput (MST) of your system. For more information about planning and measuring Maximum Sustainable Performance see Planning for Sustained Performance (http://go.microsoft.com/fwlink/?LinkId=158065) and What Is Sustainable Performance? (http://go.microsoft.com/fwlink/?LinkId=132304).

  The MST is the highest load of message traffic that a system can handle indefinitely in a production environment. All BizTalk applications should be tested for performance and throughput before going into production. At a minimum, you should run a representative set of test cases that represent the most common usage scenarios. We recommend that you test against expected loads and peak loads in a separate environment that matches characteristics of the production environment. This environment should have all of the corporate standard services installed and running, such as monitoring agents, and antivirus software.
- We also recommend that you test new BizTalk applications on the same hardware in production alongside the other BizTalk applications that are running. These other BizTalk applications put additional load on the BizTalk Server, SQL Server, network I/O, and disk I/O. In addition, one BizTalk application could cause another to throttle (when the spool depth gets too large, for example). All BizTalk applications should be performance / stress tested before going into production. In addition, you should determine how long it takes the system to recover from peak loads. If the system does not fully recover from a peak load before the next peak load occurs, then you've got a problem. The system will get further and further behind and will never be able to fully recover.

Expectations: throughput vs. latency

You can expect a certain amount of throughput and/or latency from the deployed system. Attempting to achieve high throughput and low latency simultaneously places opposing demands on the system. You can expect optimal throughput with reasonable latency. As throughput improves, stress (such as, higher CPU consumption, higher disk-I/O contention, memory pressure, and greater lock contention) on the system increases. This situation can have a negative impact on latency. To discover optimal capacity of a system, we recommend that you identify and minimize any bottlenecks.

Completed instances residing in the database can cause bottlenecks. When bottlenecks occur, performance can degrade. Giving the system sufficient time to drain can help fix the problem. Discovering the cause of the backlog buildup and helping fix the issue is important.

To discover the cause of the backlog, you can analyze historical data and monitor performance counters (to discover usage patterns, and diagnose the source of the backlog). Commonly, backlogs can occur when large volumes of data are processed in a batched manner on a nightly basis. You may find it useful to discover the capacity of the system and its ability to recover from a backlog. This information can help you to estimate hardware requirements for handling overdrive scenarios and the amount of buffer room to accommodate within a system to handle unforeseen spikes in throughput.

Monitoring performance counters can help individuate potential bottlenecks that could arise at runtime. However, when we suspect that the culprit of CPU or memory bottleneck may be one of the custom components that compose the solution, we strongly recommend that you use profiling tools such Visual Studio Profiler or ANTS Performance Profiler during a performance test to narrow and individuate with certainty the class that generates the problem. Obviously, profiling an application interferes with performance. Therefore, these tests should be focused on individuating those components that cause memory consumption, CPU utilization or latency and figures collected during these tests should be discarded.

Tuning the system for optimal sustainable throughput requires an in depth understanding of the deployed application, the strengths and weaknesses of the system, and usage patterns of the specific scenario. The only way to discover bottlenecks and predict optimal sustainable throughput with certainty is through thorough testing on a topology that closely matches what will be used in production. When running load and stress tests against a certain use case, isolating single artifacts (receive locations, send ports, orchestrations) into separate host instances and setting the right counters inside the performance monitor tool is crucial to narrow down the cause of a bottleneck.

Other topics in this section guide you through the process of defining that topology, and provide guidance on how to lessen and avoid bottlenecks.

Scaling

Bottlenecks can occur at various stages of a deployed topology. Some bottlenecks can be addressed by either scaling up or scaling out the environment. Scaling up is the process of upgrading the existing computer. For example, you can upgrade a BizTalk Server computer from a four-processor computer to an eight-processor computer, as well as substituting the existing CPUs and adding more RAM. In this way, you can accelerate resource-intensive tasks such as document parsing and mapping. Scaling out is the process of adding servers to your deployment. The decision to scale up or out depends on the type of bottleneck and the application being configured. An application needs to be built from the ground up to be capable of taking advantage of scaling up or out. The following guidance explains how to change hardware deployment topologies based on bottlenecks encountered.

Scaling up means running a BizTalk solution on upgraded hardware (for example, adding CPU's and memory). You should look at scaling up when 1) scaling out is too expensive or 2) scaling out will not help solve a bottleneck. For example, you can significantly reduce the time spent transforming and processing a large message if you run the task on a faster machine.

Scaling out the application platform consists of adding BizTalk nodes to the BizTalk Server Group and using them to run one or more parts of the solution. You should look at scaling out when 1) you need to isolate send, receive, processing, tracking hosts or 2) when memory, I/O or Network I/O resources are maxed out for a single server. The load can be spread across multiple servers; however, adding new nodes to the BizTalk Server Group can increase the lock contention on the MessageBox database.

So should you scale up or scale out? Scaling up your platform can reduce the latency of a BizTalk solution because it makes single tasks (for example, message mapping) run faster. Scaling out can improve the maximum sustainable throughtput and the scalability of your BizTalk solution because it allows you to spread the workload across multiple machines.

See Also

Finding and Eliminating Bottlenecks

System-Level Bottlenecks

This topic describes how to address common system-level bottlenecks that can impact the performance of a BizTalk Server solution.

Creating a snapshot of the baseline configuration

Gathering the following information can provide you with a snapshot of the baseline configuration that you can use to assist in correcting system-level bottlenecks.

Gather documentation

Review the documentation of the architecture and infrastructure of your scenario.

Run the Baseline Security Analyzer

Follow these steps to run the Baseline Security Analyzer:

1. Download the Microsoft Baseline Security Analyzer 2.2 (http://go.microsoft.com/fwlink/?LinkId=204006).
2. Using a domain administrator account, log on to a computer on the same network that is hosting the BizTalk Server and SQL Server computers.
3. Install the Microsoft Baseline Security Analyzer on the current computer (for example, one of the BizTalk Server computers or a separate computer).
4. Execute a separate scan (**Start Scan**), specifying as a parameter the name or the IP address of each BizTalk Server and SQL Server computer.
5. Copy each security report (.mbsa files) from the %userprofile%\SecurityScans directory.

Run the BizTalk Server Best Practices Analyzer

Follow these steps to run the BizTalk Server Best Practices Analyzer:

1. Download the BizTalk Server Best Practices Analyzer v1.2 (http://go.microsoft.com/fwlink/?LinkID=83317).

2. Using a user account that is part of the BizTalk Server Administrator security group, log on to a BizTalk Server node.
3. Install the BizTalk Server Best Practices Analyzer on the current computer.
4. Run the tool and save the report.

Run MSInfo32 and save the results

Follow these steps to run MSInfo32:

1. Using a domain or a local administrator account, log on to each BizTalk Server and SQL Server computer.
2. Launch a command prompt and change directories to the %windir%\system32 directory of the computer.
3. Run MSinfo32.exe from the command prompt.
4. Click the **File** menu and select the **Export** menu item to export to save the computer configuration.

Export the BizTalk Server and SQL Server computer TCP/IP registry setting-

Follow these steps to save the BizTalk Server and SQL Server TCP/IP registry settings:

1. Launch a command prompt and change directories to the %windir%\system32 directory of the computer.
2. Run Regedit.exe from the command prompt.
3. Navigate to the following key in the registry:

   ```
   [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters] (network settings)
   ```

4. Right-click this key and select **Export** to export the registry key to a file.

Collect the BizTalk configuration files

On each BizTalk Server node, collect the BizTalk Server configuration file, BTSNTSvc.exe.config file (or BTSNTSvc64.exe.config for 64-bit hosts), located in the BizTalk Server installation folder (e.g. C:\Program Files\Microsoft BizTalk Server 2010).

Collect the .NET configuration files

On each BizTalk Server node, collect the machine.config and web.config .NET Framework 4.0 configuration files. You can find the configuration files in the following location:

- For 32-bit: %windir%\Microsoft.NET\Framework\v4.0.30319\CONFIG folder
- For 64-bit: %windir%\Microsoft.NET\Framework64\v4.0.30319\CONFIG folder

Use the BizTalk MsgBoxViewer tool to collect information about the MessageBox database

Follow these steps to collect information about the MessageBox database using the BizTalk MsgBoxViewer tool:

1. Download the BizTalk MsgBoxViewer tool (http://go.microsoft.com/fwlink/?LinkID=117289).
2. Log on to the BizTalk Server computer with a user account that is part of the BizTalk Server Administrator security group.

3. Copy the MsgBoxViewer.exe to the BizTalk Server computer.
4. Launch the tool.
5. Click the **Optional Info to Collect** tab, and then click **Select All Info**.
6. Click **Start to Collect**.
7. When the Status label shows the **End Collection** message, switch to the folder containing the MsgBoxViewer.exe executable and copy the resulting report (.htm) and log files.

Collect and store the source code for all components used in the solution

Store the source code for all components (for example Orchestration, Custom Pipeline Component, and Helper components code) on a separate file share.

Initial troubleshooting

There are certain components of a BizTalk Server solution which, if not enabled, will cause performance problems regardless of the overall size or design of the BizTalk solution. The following preliminary troubleshooting tasks should be completed to rule out some of the "usual suspects" before engaging in an exhaustive bottleneck analysis of a BizTalk solution.

- **Verify the Tracking host instance is running -** The Tracking host instance is responsible for moving both BAM and HAT data from the TrackingData table of the MessageBox database to the BizTalkDTADb and/or BAMPrimaryImport database tables. If the tracking host instance is not running, then tracking data will accumulate in the MessageBox database and negatively impact performance of the BizTalk Server solution.
- **Verify the Enterprise Single Sign-On (ENTSSO) service is running on all BizTalk Server computers -** BizTalk host instances maintain a dependency on a locally running instance of the ENTSSO service. If the ENTSSO service is not running on the BizTalk Server, then host instances on the server will not be able to run either.
- **Verify the SQL Server Agent service is running on all SQL Server computers -** The SQL Server Agent service must be running in order for the BizTalk SQL Server Agent jobs to execute. These jobs perform important functions to keep your servers operational and healthy.
- **Verify the BizTalk SQL Server Agent jobs are enabled and running without exceptions -** Even if the SQL Server Agent service is running, it is imperative that all of the default BizTalk SQL Server Agent jobs are enabled and running successfully.
- **Check the BizTalk Server and SQL Server event logs -** A cursory examination of the BizTalk Server or SQL Server event logs may reveal a problem that could otherwise take a significant amount of time to diagnose and resolve.
- **Run the BizTalk Server Best Practices Analyzer -** The BizTalk Server Best Practices Analyzer examines a BizTalk Server deployment and generates a list of issues pertaining to best practices standards. The tool performs configuration-level verification by gathering data from different information sources, such as Windows Management Instrumentation (WMI) classes, SQL Server databases, and registry entries. The data is then used to evaluate the deployment configuration. The tool reads and reports only and does not modify any system settings, and is not a self-tuning tool. Download the BizTalk Server Best Practices Analyzer

v1.2 from BizTalk Server Best Practices Analyzer v1.2 (http://go.microsoft.com/fwlink/?LinkID=83317).

High-level system bottlenecks

This section describes system-level bottlenecks that may be present in a BizTalk Server solution and possible mitigation strategies.

Disk I/O bottlenecks

Disk I/O refers to the number of read and write operations performed by your application on a physical disk or multiple disks installed in your server. Common activities that can cause disk I/O and related bottlenecks include long-running file I/O operations, data encryption and decryption, reading unnecessary data from database tables, and an insufficient physical memory, which can lead to excessive paging activity. Disk speed is another factor to consider when evaluating disk I/O bottlenecks; faster disks provide increased performance and help reduce disk I/O bottlenecks.

The table below provides factors that should be considered when planning the disk subsystem for a BizTalk Server solution.

Disk subsystem factor	Details
Disk memory cache and available physical memory	Because memory is cached to disk as physical memory becomes limited, make sure that you have a sufficient amount of memory available. When memory is scarce, more pages are written to disk, resulting in increased disk activity. Also, make sure to set the paging file to an appropriate size. Additional disk memory cache will help offset peaks in disk I/O requests. However, it should be noted that a large disk memory cache seldom solves the problem of not having enough spindles, and having enough spindles can negate the need for a large disk memory cache. For information on configuring the Windows paging file for optimal performance, see the section “Configure the Windows PAGEFILE for optimal performance” in the topic Optimizing Operating System Performance.
Storage controller type	If you have battery-backed cache, enable write caching to improve disk write performance on the transaction log file volumes and on the database volumes. Write caching provides a response time of 2 ms for a write I/O request, as opposed to a response time of 10 to 20 ms

Disk subsystem factor	Details
	without write caching enabled. Enabling write caching greatly improves the responsiveness for client write requests. Read caching does not improve performance in a BizTalk Server scenario because it is only useful for sequential disk reads, which occur only in transaction log files. Transaction log files are read from only when they are being played back, such as after a database restore or when a server is not properly shutdown. Larger caches allow for more data to be buffered, meaning that longer periods of saturation can be accommodated. If your controller allows you to configure the cache page size, you should set it to 4 KB. A larger size, such as 8 KB, results in wasted cache because a 4 KB I/O request takes up the entire cache page of 8 KB, thereby cutting your usable cache in half.
Spindles	Spindles are more important than capacity, and BizTalk Server performance is improved if the spindles support a high number of random I/O requests. Plan for no more than 80 percent total utilization to ensure sufficient I/O is available, even in cases of a spindle failure.
Raid	The RAID solution you use should be based on the cost and performance trade-offs that are appropriate for your environment. Therefore, more than one type of RAID solution may be recommended for a particular data storage requirement. General recommendations are as follows: • Use Raid-1+0 (striped sets in a mirrored set) for the BizTalk Server databases. • Use Raid-1 (mirrored set without parity) for the transaction log file volumes. • In general, Raid 5 (striped set with distributed parity) is not recommended as Raid 5 does not provide optimal reliability, availability and performance as compared

Disk subsystem factor	Details
	to other Raid configurations. • Due to the possibility of permanent data loss, a Raid-0 (striped set without parity) configuration should never be used in a BizTalk Server production environment.
Bus type	Higher throughput provides better performance. In general, SCSI buses provide better throughput and scalability than IDE or ATA buses. You can use the following equation to determine the theoretical throughput limit for your bus type: `(Bus speed (in bits) / 8 bits per byte) X Operating speed (in MHz) = Throughput (in MB/s)` You can also improve disk I/O performance by placing multiple drives on separate I/O buses.

Performance counters for measuring disk I/O bottlenecks

Note

When attempting to analyze disk performance bottlenecks, you should always use physical disk counters. However, if you use software RAID, you should use logical disk counters. As for Logical Disk and Physical Disk Counters, the same counters are available in each of these counter objects. Logical disk data is tracked by the volume manager (or managers), and physical disk data is tracked by the partition manager.

The following performance counters should be used to determine if your system is experiencing a disk I/O related bottleneck:

- **PhysicalDisk\Avg. Disk Queue Length**
  - Threshold: Should not be higher than the number of spindles plus two.
  - Significance: This counter indicates the average number of both read and writes requests that were queued for the selected disk during the sample interval. This counter is useful for gathering concurrency data, including data bursts and peak loads. These values represent the number of requests in flight below the driver taking the statistics. This means the requests are not necessarily queued but could actually be in service or completed and on the way back up the path. Possible in-flight locations include the following:
    - SCSIport or Storport queue
    - OEM driver queue
    - Disk controller queue
    - Hard disk queue

    - Actively receiving from a hard disk
- **PhysicalDisk\Avg. Disk Read Queue Length**

  Threshold: Should be less than two.

  Significance: This counter indicates the average number of read requests that were queued for the selected disk during the sample interval.
- **PhysicalDisk\Avg. Disk Write Queue Length**
  - Threshold: Should be less than two.
  - Significance: This counter indicates the average number of write requests that were queued for the selected disk during the sample interval.
- **PhysicalDisk\Avg. Disk sec/Read**
  - Threshold: No specific value.
    - Less than 10 milliseconds (ms) = good
    - Between 15 and 25 ms = fair
    - Greater than 25 ms = poor
  - Significance: This counter indicates the average time, in seconds, of a data read operation from the disk. If the number is larger than 25 milliseconds (ms), that means the disk system is experiencing latency when reading from the disk. For mission-critical servers hosting BizTalk Server, the acceptable threshold is much lower, approximately 10 ms.
- **PhysicalDisk\Avg. Disk sec/Write**
  - Threshold: No specific value.
    - Less than 10 milliseconds (ms) = good
    - Between 15 and 25 ms = fair
    - Greater than 25 ms = poor
  - Significance: This counter indicates the average time, in seconds, of a data write operation to the disk. If the number is larger than 25 ms, the disk system experiences latency when writing to the disk. For mission-critical servers hosting BizTalk Server, the acceptable threshold is much lower, approximately 10 ms.
- **PhysicalDisk\Avg. Disk sec/Transfer**
  - Threshold: Should not be more than 18 milliseconds.
  - Significance: This counter indicates the time, in seconds, of the average disk transfer. This may indicate a large amount of disk fragmentation, slow disks, or disk failures. Multiply the values of the **Physical Disk\Avg. Disk sec/Transfer** and **Memory\Pages/sec** counters. If the product of these counters exceeds 0.1, paging is taking more than 10 percent of disk access time, so you need more physical memory available.
- **PhysicalDisk\Disk Writes/sec**
  - Threshold: Depends on manufacturer's specifications.
  - Significance: This counter indicates the rate of write operations on the disk.
- **Processor\% DPC Time, % Interrupt Time and % Privileged Time -** If Interrupt Time and Deferred Procedure Call (DPC) time are a large portion of Privileged Time, the kernel is

spending a significant amount of time processing I/O requests. In some cases performance can be improved by configuring interrupts and DPC affinity to a small number of CPUs on a multiprocessor system, which improves cache locality. In other cases, it works best to distribute the interrupts and DPCs among many CPUs, so as to keep the interrupt and DPC activity from becoming a bottleneck. For information on using the Interrupt Filter Configuration tool to bind network adapter interrupts to specific processors on multiprocessor computers, see the section "Use the Interrupt Filter Configuration tool to bind network adapter interrupts to specific processors on multiprocessor computers" in Optimizing Operating System Performance.

- **Processor\DPCs Queued / sec -** Measures how DPCs are consuming CPU time and kernel resources.
- **Processor\Interrupts / sec -** Another measurement of how interrupts are consuming CPU time and kernel resources. Modern disk controllers often combine or coalesce interrupts so that a single interrupt results in the processing of multiple I/O completions. Of course, there is a trade-off between delaying interrupts (and thus completions) and economizing CPU processing time.

Disk I/O tuning options

If you determine that disk I/O is a bottleneck in your environment, the following techniques may be used to alleviate the bottleneck:

- **Defragment your disks -** Use the available at PageDefrag utility (http://go.microsoft.com/fwlink/?LinkId=108976) to defragment the Windows paging file and pre-allocate the Master File Tables.
- **Use stripe sets to process I/O requests concurrently over multiple disks -** Use mirrored volumes to provide fault tolerance and increase I/O performance. If you do not require fault tolerance, implement stripe sets for fast reading and writing and improved storage capacity. When stripe sets are used, per disk utilization is reduced because work is distributed across the volumes, and overall throughput increases. If you adding additional disks in a stripe set does not increase throughput, then your system might be experiencing a bottleneck due to contention between disks by the disk controller. In this case, adding an additional disk controller will help distribute load and increase performance.
- **Distribute workload among multiple drives -** Windows Clustering and Distributed File System provide solutions for load balancing on multiple disk drives.
- **Limit the use of file compression or encryption -** File compression and encryption are I/O-intensive operations. You should only use them where absolutely necessary.
- **Disable creation of short names -** If you are not supporting MS-DOS for Windows 3.x clients, disable short names to improve performance. For more information about disabling the creation of short names, see the section "Disable short-file-name (8.3) generation" in Optimizing Operating System Performance.
- **Disable last access update -** By default, NTFS updates the date and time stamp of the last access on directories whenever it traverses the directory. For a large NTFS volume, this update process may hinder performance. For more information about disabling last access updates, see the section "Disable NTFS last access updates" in Optimizing Operating System Performance.

Caution

Some applications, such as incremental backup utilities, rely on the NTFS update information and will not function correctly without it.

- **Reserve appropriate space for the master file table -** Add the **NtfsMftZoneReservation** entry to the registry depending on the number of files that are typically stored on your NTFS volumes. When you add this entry to the registry, the system reserves space on the volume for the master file table. Reserving space in this manner allows the master file table to grow optimally. If your NTFS volumes generally store relatively few files, set the value of this registry entry to the default value of 1. Typically you can use a value of 2 or 3 if your NTFS volumes store a moderate numbers of files, and use a value of 4 (the maximum) if your NTFS volumes tend to contain a relatively large number of files. However, make sure to test any settings greater than 2, because these greater values cause the system to reserve a much larger portion of the disk for the master file table. For more information about adding the **NtfsMftZoneReservation** to the registry, see the section "Increase space available for the master file table" in Optimizing Operating System Performance.
- **Use the most efficient disk systems available -** In addition to the physical disk that is used, consider the type of disk controller and cabling that will be used. An efficient disk subsystem should also provide drivers that support interrupt moderation or interrupt avoidance to mitigate processor interrupt activity caused by disk I/O.
- **Ensure that you are using the appropriate RAID configuration -** Use RAID 10 (striping and mirroring) for optimal performance and fault tolerance. The tradeoff is that using RAID 10 is expensive. Avoid using RAID 5 when you have extensive write operations. For more information about implementing RAID in a BizTalk Server environment, see the section "Disk Infrastructure" in the BizTalk Server Database Optimization white paper (http://go.microsoft.com/fwlink/?LinkID=101578).
- **Consider using database partitions -** If you have a database bottleneck, consider using database partitions and mapping disks to specific tables and transaction logs. The primary purpose of partitions is to overcome disk bottlenecks for large tables. If you have a table with large number of rows and you determine that it is the source of a bottleneck, consider using partitions. For SQL Server, you can use file groups to improve I/O performance. You can associate tables with file groups, and then associate the file groups with a specific hard disk. For more information about using optimizing filegroups for the BizTalk Server databases, see Optimizing Filegroups for the Databases.
- **Consider adding physical memory, if you have excessive page faults -** A high value for the **Memory: Pages/sec** performance counter could indicate excessive paging which will increase disk I/0. If this occurs, consider adding physical memory to reduce disk I/O and increase performance.
- **Consider using a disk with a higher RPM rating or using a Storage Area Network (SAN) device -** Disks with higher RPM ratings offer improved performance compared to disks with lower RPM ratings. SAN devices typically offer top tier performance but at a price premium.
- Follow the recommendations in Optimizing Database Performance. This topic provides several recommendations for optimizing database performance both before and after configuring BizTalk Server.

CPU bottlenecks

Each application that runs on a server gets a time slice of the CPU. The CPU might be able to efficiently handle all of the processes running on the computer, or it might be overloaded. By examining processor activity and the activity of individual processes including thread creation, thread switching and context switching, you can gain insight into processor workload and performance.

You may have a CPU bottleneck if…

- The value of the **Processor\% Processor Time** performance counter often exceeds 75%.
- The value of the **System\ Processor Queue Length** performance counter is 2 or more for a sustained period of time.
- The value of the **Processor\% Privileged Time** or **System\Context Switches/sec** performance counters are unusually high.

  If the value of **Processor\ % Processor Time** performance counter is high, then queuing occurs, and in most scenarios the value of **System\ Processor Queue Length** will also be high.

Performance counters for measuring CPU bottlenecks

The following performance counters should be used to determine if your system is experiencing a CPU related bottleneck:

- **Processor\% Processor Time**

  Threshold: Should be less than 85%.

  Significance: This counter is the primary indicator of processor activity. High values many not necessarily be bad. However, if the other processor-related counters are increasing linearly such as **Processor\% Privileged Time** or **System\Processor Queue Length**, high CPU utilization may be worth investigating.

- **Processor\% Privileged Time**

  Threshold: A figure that is consistently over 75 percent indicates a bottleneck.

  Significance: This counter indicates the percentage of time a thread runs in privileged mode. When your application calls operating system functions (for example to perform file or network I/O or to allocate memory), these operating system functions are executed in privileged mode.

- **Processor\% Interrupt Time**

  Threshold: Depends on the processor.

  Significance: This counter indicates the percentage of time the processor spends receiving and servicing hardware interrupts. This value is an indirect indicator of the activity of devices that generate interrupts, such as network adapters. A dramatic increase in this counter indicates potential hardware problems.

- **System\Processor Queue Length**

  Threshold: An average value consistently higher than 2 indicates a bottleneck.

  Significance: If there are more tasks ready to run than there are processors, threads queue up. The processor queue is the collection of threads that are ready but not able to be

executed by the processor because another thread is currently executing. A sustained or recurring queue of more than two threads is a clear indication of a processor bottleneck. You may get more throughput by reducing parallelism in those cases.

You can use this counter in conjunction with the **Processor\% Processor Time** counter to determine if your application can benefit from more CPUs. There is a single queue for processor time, even on multiprocessor computers. Therefore, in a multiprocessor computer, divide the Processor Queue Length (PQL) value by the number of processors servicing the workload.

If the CPU is very busy (90 percent or higher utilization) and the PQL average is consistently higher than 2 per processor, you may have a processor bottleneck that could benefit from adding CPUs. Or, you could reduce the number of threads and queue more at the application level. This will cause less context switching, and less context switching is good for reducing CPU load. A common reason for a PQL value of 2 or higher with low CPU utilization is that requests for processor time arrive randomly and threads demand irregular amounts of time from the processor. This means that the processor is not a bottleneck but that the application threading logic should be improved.

- **System\Context Switches/sec**

  Threshold: As a general rule, a context switching rate less than 5,000 per second per processor is not worth worrying about. If context switching rates exceed 15,000 per second per processor, then context switching can become a bottleneck.

  Significance: Context switching occurs when a higher priority thread preempts a lower priority thread that is currently running or when a high priority thread blocks other threads. High levels of context switching can occur when many threads share the same priority level. This often indicates that there are too many threads competing for the processors on the system. If both processor utilization and context switching levels are low, this is often an indication that threads are blocked.

Resolving CPU bottlenecks

The first step is to identify which process is consuming excessive processor time. Use Windows **Task Manager** to identify which process is consuming high levels of CPU by looking at the **CPU** column on the **Processes** page. You can also determine this by monitoring **Process\%Processor Time** in Performance Monitor and selecting the processes you want to monitor.

Another powerful tool for determining which processes are consuming excessive CPU time is the Visual Studio Team System Profiler that is available with the Visual Studio Team System (VSTS) suite. For more information about using the Visual Studio Team System Profiler, see the Visual Studio Team System documentation.

After you determine that your CPU is a bottleneck you have several options, including the following:

- Add multiple processors if you have multi-threaded applications. Consider upgrading to a more powerful processor if your application is single-threaded.

- If you observe a high rate of context switching, consider reducing the thread count for your process before increasing the number of processors.
- Analyze and tune the application that is causing high CPU utilization. You can dump the running process by using the ADPLUS utility and analyze the cause by using Windbg. These utilities are part of the Windows debugging toolkit. You can download these tools from Debugging Tools for Windows (http://go.microsoft.com/fwlink/?LinkID=106624).
- Analyze the instrumentation log generated by your application to isolate the subsystem that is taking the maximum amount of time for execution. Determine whether a code review would be more beneficial than just tuning the BizTalk Server environment.

Note

Although you can change the process priority level of an application by using **Task Manager** or from a command prompt, you should generally avoid doing so.

Memory bottlenecks

When evaluating memory-related bottlenecks, consider unnecessary allocations, inefficient clean up, and inappropriate caching and state management mechanisms. To resolve memory-related bottlenecks, optimize your code to eliminate these issues and then tune the amount of memory allocated to your application(s). If you determine during tuning that memory contention and excessive paging are occurring, you may need to add additional physical memory to the server. Low memory leads to increased paging of an application's virtual address space to and from disk. If paging becomes excessive, disk I/O will increase and negatively impact overall system performance.

You might have a memory bottleneck if…

- The value of the **Memory\Available MBytes** performance counter is low, either due to system memory limitations or an application that is not releasing memory. Monitor the value of the **Process\Working Set** performance counter for each process that is running. If value of the **Process\Working Set** remains high even when the process is not active, this may be an indication that the process is not releasing memory as it should.
- The value of the **Memory\Pages/sec** performance counter is high. The average of **Memory\Pages Input/sec** divided by average of **Memory\Page Reads/sec** gives the number of pages per disk read. This value should not generally exceed five pages per second. A value greater than five pages per second indicates that the system is spending too much time paging and requires more memory (assuming that the application has been optimized).

Performance counters for measuring memory bottlenecks

The following performance counters should be used to determine if your system is experiencing a memory related bottleneck:

- **Memory\Available Mbytes**

  Threshold: A consistent value of less than 20 to 25 percent of installed RAM is an indication of insufficient memory.

Significance: This indicates the amount of physical memory available to processes running on the computer. Note that this counter displays the last observed value only. It is not an average.

- **Memory\Page Reads/sec**

  Threshold: Sustained values of more than five indicate a large number of page faults for read requests.

  Significance: This counter indicates that the working set of a process is too large for the available physical memory which is causing memory to page to disk. It shows the number of read operations, without regard to the number of pages retrieved in each operation. Higher values indicate a memory bottleneck.

  If a low rate of page-read operations coincides with high values for **Physical Disk\% Disk Time** and **Physical Disk\Avg. Disk Queue Length**, a disk I/O bottleneck condition may exist. If an increase in queue length is not accompanied by a decrease in the pages-read rate, a memory bottleneck exists due to insufficient physical memory.

- **Memory\Pages/sec**

  Threshold: A sustained value of more than five indicates a bottleneck.

  Significance: This counter indicates the rate at which pages are read from or written to disk to resolve hard page faults. Multiply the values of the **Physical Disk\Avg. Disk sec/Transfer** and **Memory\Pages/sec** performance counters. If the product of these values exceeds **0.1**, paging is utilizing more than 10 percent of disk access time, which indicates that insufficient physical memory is available.

- **Memory\Pool Nonpaged Bytes**

  Threshold: Watch the value of **Memory\Pool Nonpaged Bytes** for an increase of 10 percent or more from its value at system startup.

  Significance: An increase of 10 percent or more from the value at startup may be an indication of a memory leak.

- **Server\Pool Nonpaged Failures**

  Threshold: Regular nonzero values indicate a bottleneck.

  Significance: This counter indicates the number of times allocations from the non-paged pool have failed. The non-paged pool contains pages from a process's virtual address space that are not to be swapped out to the page file on disk, such as a process kernel object table. The availability of the non-paged pool determines how many processes, threads, and other such objects can be created. When allocations from the non-paged pool fail, this can be due to a memory leak in a process, particularly if processor usage has not increased accordingly.

- **Server\Pool Paged Failures**

  Threshold: No specific value.

  Significance: This counter indicates the number of times allocations from the paged pool have failed. A positive value for this counter is an indication that the computer has insufficient physical memory or that the Windows paging file is too small.

- **Server\Pool Nonpaged Peak**

  Threshold: No specific value.

  Significance: This is the maximum number of bytes in the non-paged pool that the server has reserved for use at any one point. It indicates how much physical memory the computer should have. Because the non-paged pool must be resident in physical memory and because there has to be some memory left over for other operations, as a rule of thumb the computer should have installed physical memory that is 4 times the value indicated for this counter.

- **Memory\Cache Bytes**

  Threshold: No specific value.

  Significance: Monitors the size of cache under different load conditions. This value of this performance counter indicates the size of the static files cache. By default, this counter uses approximately 50 percent of available memory, but decreases if available memory shrinks, which affects system performance.

- **Memory\Cache Faults/sec**

  Threshold: No specific value.

  Significance: This counter indicates how often the operating system looks for data in the file system cache but fails to find it. This value should be as low as possible.

- **Cache\MDL Read Hits %**

  Threshold: The higher this value, the better the performance of the file system cache. Values should be as close to 100 percent as possible.

  Significance: This counter provides the percentage of Memory Descriptor List (MDL) Read requests to the file system cache, where the cache returns the object directly rather than requiring a read from the hard disk.

- **Process\Working Set**

  Threshold: No specific value.

  Significance: The working set is the set of memory pages currently loaded in memory (physical + virtual). If the system has sufficient memory, it can maintain enough space in the working set so that it does not need to perform disk operations to page memory to disk. However, if there is insufficient memory, the system tries to reduce the working set by taking away the memory from the processes which results in an increase in page faults. When the rate of page faults rises, the system tries to increase the working set of the process. If you observe wide fluctuations in the working set, it might indicate a memory shortage. Higher values in the working set may also be due to multiple assemblies in an application. You can improve the working set by using assemblies shared in the global assembly cache.

Resolving memory bottlenecks

If you determine that memory is a bottleneck in your BizTalk Server environment, use one or more of the following methods to resolve the bottleneck:

- Tune the amount of memory allocated if you can control the allocation. For example, you can tune this for BizTalk Server, ASP.NET and SQL Server.
- Increase the size of the Windows paging file and follow the steps in the “Configure the Windows PAGEFILE for optimal performance” section of Optimizing Operating System Performance.
- Turn off services that are not used. Stopping services that you do not use regularly saves memory and improves system performance. For more information, see the “Disable non-essential services” section of Optimizing Operating System Performance.
- Remove unnecessary protocols and drivers. Even idle protocols use space in the paged and non-paged memory pools. Drivers also consume memory, so you should remove unnecessary ones. For more information, see the “Remove unnecessary network protocols” section of Optimizing Operating System Performance.
- Install additional physical memory on the computers in the BizTalk Server environment.

Note

On a 32-bit system, BizTalk can use a maximum of 2GB of memory, the limit increases to 3GB with BizTalk Server 2010 and later if the /3GB switch is used. For more information on memory usage, see Memory Limits for Windows Releases (http://go.microsoft.com/fwlink/?LinkId=118349).

Network I/O bottlenecks

You might have a network I/O bottleneck if…

- The value of the **Network Interface\Bytes Total/sec** performance counter exceeds 80% of available network bandwidth.
- The value of the **Server\Bytes Total/sec** performance counter is greater than 50% of available network bandwidth.

Performance counters for measuring network I/O bottlenecks

The following performance counters should be used to measure network I/O and determine if your system is experiencing a network I/O related bottleneck:

- **Network Interface\Bytes Total/sec**

  Threshold: Sustained value of more than 80 percent of network capacity.

  Significance: This counter indicates the rate at which bytes are sent and received over each network adapter. This counter helps determine if a network adapter is saturated and whether you should add one or more network adapters to increase available network bandwidth.

- **Network Interface\Bytes Received/sec**

  Threshold: No specific value.

  Significance: This counter indicates the rate at which bytes are received over each network adapter. Use the value of this counter to calculate the rate of incoming data as a percentage of total available bandwidth. This will help determine if network bandwidth

should be increased on the client sending data to BizTalk Server or if network bandwidth should be increased on the BizTalk Server computer itself.

- **Network Interface\Bytes Sent/sec**

  Threshold: No specific value.

  Significance: This counter indicates the rate at which bytes are sent over each network adapter. Use the value of this counter to calculate the rate of outgoing data as a percentage of total available bandwidth. This will help determine if network bandwidth should be increased on the BizTalk Server sending data to a client or if network bandwidth should be increased on the client computer receiving data from BizTalk Server.

- **Server\Bytes Total/sec**

  Threshold: Sustained value of more than 50 percent of network capacity.

  Significance: This counter indicates the number of bytes sent and received over the network. Higher values indicate a network I/O bottleneck. If the sum of **Bytes Total/sec** for all servers is roughly equal to the maximum transfer rate of your network, consider subnetting the network to improve performance. For more information about subnetting a network to improve performance, see the **Subnets** section of the BizTalk Server Database Optimization white paper (http://go.microsoft.com/fwlink/?LinkID=101578).

Resolving network I/O bottlenecks

If you determine that network I/O is a bottleneck in your environment, use one or more of the following methods to resolve the bottleneck:

- Follow the recommendations in the “General guidelines for improving network performance” section of Optimizing Network Performance.
- Run the Windows Powershell network optimization script described in the topic Optimizing Network Performance on each computer in the BizTalk Server environment.

See Also

Finding and Eliminating Bottlenecks

Bottlenecks in the BizTalk Server Tier

The BizTalk tier can be divided into the following functional areas:

- Receiving
- Processing
- Transmitting
- Tracking
- Other

For these areas, if the system resources (CPU, memory, and disk) appear to be saturated, upgrade the server by scaling the platform up or out based on the characteristics of your application. If the system resources are not saturated, perform the steps described in this section.

Bottlenecks in the receive location

If messages build up at the receive location (for example, file receive folder grows large), this indicates that the system is unable to absorb data fast enough to keep up with the incoming load. This is due to internal throttling. That is, BizTalk Server reduces the receiving rate if the subscribers are unable to process data fast enough causing backlog buildup in the database tables. If the bottleneck is caused by hardware limitations, try scaling up. For more information about scaling up, see the following topics in the BizTalk Server 2010 documentation:

- Scaling Up the BizTalk Server Tier (http://go.microsoft.com/fwlink/?LinkId=158073).
- Scaling Up the SQL Server Tier (http://go.microsoft.com/fwlink/?LinkId=158077).

It is also possible to scale out by adding a host instance (server) to the host mapped to the receive handler. For more information about scaling out, see the following topics in the BizTalk Server 2010 documentation:

- Scaling Out the BizTalk Server Tier (http://go.microsoft.com/fwlink/?LinkId=158076).
- Scaling Out the SQL Server Tier (http://go.microsoft.com/fwlink/?LinkId=158075).

Use Perfmon to monitor the resource use on the system. It is important to confirm that the external receive location is not the cause of the bottleneck. For example, confirm whether the remote file share is saturated due to high disk I/O, the server hosting the remote outgoing queue is not saturated, or the client used to generate HTTP load is not starved on threads.

Processing bottlenecks

If the Host Queue – Length performance counter is climbing, it indicates that the orchestrations are not completing fast enough. For more information, see the Perfmon counter table in this topic. This could be due to memory contention or CPU saturation.

If the orchestration servers are the bottleneck, use Perfmon to identify the source.

If the server is CPU bound, consider the following:

- If the workflow is complex, consider splitting the orchestration into multiple smaller orchestrations

  Note
  Splitting an orchestration into multiple workflows can cause additional latency and add complexity. Multiple workflows can also cause an increase in the number of messages published to and consumed from the BizTalkMsgBoxDb, putting additional pressure on the database.

- If you use complex maps, consider whether they can be moved to the Receive/Send ports. Be sure to verify which ports have additional bandwidth.
- Consider scaling up the hardware or scaling out by configuring an additional processing server.

Transmitting bottlenecks

If the server hosting the send adapters is saturated on resources (for example, disk, memory, or CPU), consider scaling-up the server or scaling-out to additional send host servers. The sending

tier could become the bottleneck if the destination (external to BizTalk) is unable to receive data fast enough. This will cause messages to buildup in the MessageBox database (Application SendHostQ).

If all the endpoints are within the scope of the topology, isolate the cause at the destination. For example, determine if the HTTP location is optimally configured to receive load. If not, consider scaling out. Also determine if the destination is growing due to excessive output messages delivered by BizTalk. If yes, you might need a maintenance plan to archive and purge the destination messages. Large numbers of files in a destination folder can severely impact the ability of the BizTalk service to commit data to the disk drive.

Tracking bottlenecks

The Tracking host instance is responsible for moving the Business Activity Monitoring (BAM) and Health and Activity Tracking (HAT) data from the MessageBox database (TrackingData table) to the BizTalk Tracking and/or BAM Primary Import database tables. If multiple MessageBox databases are configured, the tracking host instance uses four threads per MessageBox database.

It is possible that the Tracking host instance is CPU bound. If it is, consider scaling-up the server or scale-out by configuring an additional server with Host Tracking enabled. The multiple host instances will automatically load balance for the multiple MessageBox databases configured. For more information about scaling, see the topic Scaling Your Solutions (http://go.microsoft.com/fwlink/?LinkId=158078).

If the TrackingData table in the MessageBox database grows large, it is usually because the data maintenance jobs on the BizTalk Tracking and/or BAM Primary Import database are not running as configured, causing growth of the BizTalk Tracking and/or BAM Primary Import databases. After these databases grow too large it can have a negative impact on the ability of the Tracking host to insert data into the TrackingData table. This causes tracked data to back up in the MessageBox database tables. The growth of the TrackingData table cause throttling to start.

You should only enable the minimum tracking required for your application, as this will reduce the amount of data logged and lower the risk of tracking bottlenecks. For information about disabling tracking settings for individual items such as orchestrations and send/receive ports, see How to Disable Tracking (http://go.microsoft.com/fwlink/?LinkId=160064).

Other

Configure the deployment topology such that different functionality runs in dedicated isolated host instances. This way each host instance gets its own set of resources (for example, on a 32-bit system, 2GB virtual memory address space, handles, threads). If the server is has sufficient CPU headroom and memory to host multiple host instances, they can be configured to run on the same physical computer. If not, consider scaling out by moving the functionality to dedicated servers. Running the same functionality on multiple servers also serves to provide a highly available configuration.

BizTalk system performance counters

Object	Instance	Counter	Monitoring Purpose
Processor	_Total	% Processor Time	Resource Contention
Process	BTSNTSvc	Virtual Bytes	Memory Leak/Bloat
Process	BTSNTSvc	Private Bytes	Memory Leak/Bloat
Process	BTSNTSvc	Handle Count	Resource Contention
Process	BTSNTSvc	Thread Count	Resource Contention
Physical Disk	_Instance	% Idle Time	Resource Contention
Physical Disk	_Instance	Average Disk Queue Length	Resource Contention

CPU contention

If the processor is saturated, you can fragment the application by separating the receiving from the sending and orchestration. To do this, create separate hosts, map the hosts to specific functionality (receive/send/orchestrations/tracking) and add dedicated servers to these separate hosts. Orchestration functionality is often CPU-intensive. If you configure the system so the orchestrations execute on a separate dedicated server, this can help improve overall system throughput.

If multiple orchestrations are deployed, you can enlist them to different dedicated orchestration hosts. Mapping different physical servers to the dedicated orchestration hosts ensures that the different orchestrations are isolated and do not contend for shared resources either in the same physical address space or on the same server.

Stop unused host instances. Unused host instances can compete for CPU and memory resources by regularly checking the MessageBox for messages to process. Additionally, stop unused receive locations, send ports, and orchestrations.

Spool table growth

Downstream bottlenecks and/or resource contention can cause the spool to start growing excessively and reduce overall performance. For more information, see “Spool Table Growth” in How to Identify Bottlenecks in the MessageBox Database.

Memory starvation

High throughput scenarios can have increased demand on system memory. Since a 32-bit process is limited by the amount of memory it can consume, it is recommended to separate the receive/process/send functionality into separate host instances such that each host receives its own 2GB address space. In addition, if multiple host instances are running on the same physical server, you can upgrade to 4/8GB memory to avoid swapping data to disk from real memory.

Long running orchestrations can hold onto allocated memory longer. This can cause memory bloat and throttling to start. Large messages can also cause high memory consumption.

You can ease the memory bloat problem that occurs when large messages are processed by lowering the **Internal Message Queue Size** and **In-process Messages per CPU** values for the specific host.

Note
If latency is a concern, changes to **Internal Message Queue Size** and **In-process Messages per CPU** should be made with caution as this may increase end-to-end latency of the system.

Disk contention

If the disks are saturated (for example, with a large number of FILE/MSMQ transports), consider upgrading to multiple spindles and striping the disks with RAID 10. In addition, whenever using the FILE transport, it is important to ensure that the receive and send folders do not grow larger than 50,000 files.

The receive folder can grow large if BizTalk Server throttles incoming data into the system. It is important to move data from the send folder so that growth in this folder does not impact the ability of BizTalk Server to write additional data. For non-transactional MSMQ queues, it is recommended to remotely create the receive queues so that disk contention is reduced on the BizTalk Server.

The remote non-transactional queue configuration also provides high availability as the remote server hosting the queue can be clustered.

Other system resource contention

Depending on the type of transport, it may be necessary to configure system resources like IIS for HTTP (for example, MaxIOThreads, MaxWorkerThreads).

With ASP.NET 2.0 and ASP.Net 4, the <processModel autoConfig="true"/> setting in the machine.config file will automatically configure the following settings for optimal performance:

- Maximum Worker Threads
- Maximum IO Threads
- minFreeThreads attribute of the httpRuntime element
- minLocalRequestFreeThreads attribute of the httpRuntime element
- maxConnection attribute of the <connectionManagement> Element (Network Settings) element

For more information about configuring parameters that affect adapter performance, see ASP.NET Configuration Settings for the processModel Element (http://go.microsoft.com/fwlink/?LinkId=158080).

For more information about configuration settings that can affect BizTalk Server adapters, see Configuration Parameters that Affect Adapter Performance (http://go.microsoft.com/fwlink/?LinkID=154200).

In addition to optimizing your BizTalk Server applications, other ASP.NET applications running on the same server can have an affect on overall performance. It is important to optimize all of your ASP.NET applications to reduce demand on system resources. For more information, see ASP.NET Performance (http://go.microsoft.com/fwlink/?LinkId=158081).

When configuring for maximum performance, consider optimizing other external systems such as messaging engines (Message Queuing, MQSeries), applications, databases, Active Directory, etc. that are part of your overall BizTalk solution. Performance bottlenecks in any of these other external systems can have a negative effect on your BizTalk solution.

Downstream bottlenecks

If the downstream system is unable to receive data fast enough from BizTalk, this output data will back up in the BizTalk databases. This results in bloat, causes throttling to start, shrinks the receive pipe, and impacts the overall throughput of the BizTalk system. A direct indication of this is Spool table growth. For more information about bottlenecks and the Spool table, see How to Identify Bottlenecks in the MessageBox Database.

Throttling impact

Throttling will eventually start to protect the system from reaching an unrecoverable state. Thus, you can use throttling to verify whether the system is functioning normally and discover the source of the problem. After you identify the cause of the bottleneck from the throttling state, analyze the other performance counters to determine the source of the problem. For example, high contention on the MessageBox database could be due to high CPU use, caused by excessively paging to disk that is caused by low memory conditions. High contention on the MessageBox database could also be caused by high lock contention due to saturated disk drives. While occasional throttling is not a significant threat to performance, persistent throttling can indicate a more significant underlying problem. For more information about those conditions where BizTalk Server will use throttling, see How BizTalk Server Implements Host Throttling (http://go.microsoft.com/fwlink/?LinkID=155286).

For more information about how BizTalk Server throttling can help manage the use of available resources and minimize resource contention, see Optimizing Resource Usage Through Host Throttling (http://go.microsoft.com/fwlink/?LinkID=155770).

BizTalk application counters

Object	Instance	Counter	Description
BizTalk Messaging	RxHost	Documents Received/Sec	Incoming Rate
BizTalk Messaging	TxHost	Documents Processed/Sec	Outgoing Rate
XLANG/s Orchestrations	PxHost	Orchestrations Completed/Sec.	Processing Rate

Object	Instance	Counter	Description
BizTalk : MessageBox: General Counters	MsgBoxName	Spool Size	Cumulative size of all Host Queues
BizTalk : MessageBox: General Counters	MsgBoxName	Tracking Data Size	Size of TrackingData table on the MessageBox
BizTalk:MessageBox:Host Counters	PxHost:MsgBoxName	Host Queue - Length	Number of messages in the specific Host Queue
BizTalk:MessageBox:Host Counters	TxHost:MsgBoxName	Host Queue - Length	Number of messages in the specific Host Queue
BizTalk:Message Agent	RxHost	Database Size	Size of publishing (PxHost) Queue
BizTalk:Message Agent	PxHost	Database Size	Size of publishing (TxHost) Queue
BizTalk:Message Agent	HostName	Message Delivery Throttling State	Affects XLANG and Outbound transports
BizTalk:Message Agent	HostName	Message Publishing Throttling State	Affects XLANG and Inbound transports

Where do I start?

Monitoring the **Message Delivery Throttling State** and the **Message Publishing Throttling State** for each host instance is a good place to start. If the value of these counters is not zero, this indicates throttling in the BizTalk system and you can further analyze the cause of the bottleneck. For descriptions on the other performance counters, see Bottlenecks in the Database Tier.

Orchestration engine performance counters

We strongly recommend that you fine tune orchestration runtime settings because continuous orchestration dehydration\rehydration and persistence points can have a severe impact on overall performance. You must use the following counters when testing complex orchestrations that may contain multiple persistence points and transactional scopes.

Counter	Comments
Orchestrations dehydrated/sec	Average number dehydrated per second.
Orchestrations rehydrated/sec	Average number rehydrated per second.
Persistence points/sec	Average number of orchestration instances persisted per second.
Orchestrations resident in memory	Number of orchestration instances currently hosted by the host instance.
Orchestrations completed/sec	Average number completed per second.
Orchestrations suspended/sec	Average number suspended per second.
Transactional scopes committed/sec	Average number committed per second.
Transactional scopes aborted/sec	Average number aborted per second.
Transactional scopes compensated/sec	Average number compensated per second.

Backlog buildup

For a 1-1 deployment scenario where 1 message received results in 1 message processed and transmitted, if the outgoing rate does not equal the incoming rate, there is a backlog in the system. In this situation, you can monitor the Spool Size. When determining the cause of bottlenecks in outgoing rate, run a single use-case scenario at a time. Isolate orchestrations, receive locations, and send locations to separate hosts.

Add the BizTalk:Message Box:Host Counters to your Performance Monitor log to monitor a host. The Host Queue - Length: counter tracks the total number of messages in a particular host queue. If one or more of these counters continues grow over time, give particular attention to the artifacts executed by those hosts.

If the Spool is growing linearly, determine which Application Queue is responsible for the Spool growth.

If none of the Application Queues are growing and the Spool continues to grow, it could mean that the purge jobs are unable to keep up. This occurs if the agent is not running or there is other system resource contention on the SQL Server.

If one of the Application Queues is growing, diagnose the cause of this growth. Monitor the system resources on the system that is unable to drain the specific Application Queue (for

example, Orchestration Host-Q is growing due to CPU starvation on the server). In addition, verify the values of the throttling counter for the specific host instance.

If the BizTalk:Messsage Agent Message Delivery Throttling State and Message Publishing Throttling State performance counters are not zero, check the value to confirm the reason for throttling (for example, memory threshold exceeded, in-flight message count too high, and so on).

Stand-Alone Profiler

You can use performance counters to detect the location of the bottleneck at a high level. However, once narrowed down, you might need to examine the code more closely to help ease the problem. The Stand-Alone Profiler that ships with Visual Studio 2010 can be a very helpful tool to help diagnose where the code is spending most of its cycles. For more information, see

L2/L3 cache

From a hardware perspective, you can gain the biggest benefits by using the onboard CPU cache. Higher CPU cache helps increase cache hit rate reducing the need for the system to page data in and out of memory to disk.

64-Bit performance bottlenecks

Performance on 64-bit systems may appear lower than what can be achieved on 32-bit systems. This is possible for a few reasons, the most important one being memory.

Measuring performance on a 32-bit system with 2 GB of memory and comparing the results to a similar 64-bit system with 2 GB of memory is not comparing the same thing. The 64-bit system will appear to be disk-I/O bound (low % Disk Idle time & high Disk Queue Length) and CPU bound (max CPU and high context switching). However, this is not because performing file I/O on a 64-bit system is more expensive.

The 64-bit system is more memory intensive (64-bit addressing) which results in the operating system consuming most of the 2 GB available memory. When this happens, most other operations cause paging to disk which stresses the file subsystem. Therefore, the system spends CPU cycles paging in/out of memory both data and code and is impacted by the high disk latency cost. This manifests itself as both higher disk contention and higher CPU consumption.

The way to alleviate this problem is to scale-up the server by upgrading the memory. Scaling-up to 8 GB is idea, however, adding more memory will not help improve throughput unless the source of the problem is CPU starvation due to low memory conditions.

Using BAM to identify bottlenecks and high-latency issues

When low-latency is important, you can use BAM to measure the time the system takes to complete each stage within the BizTalk Server system. Although you can use HAT to debug the state of messages and diagnose the source of problems in routing messages, you can use BAM to track various points through the message flow. By creating a BAM tracking profile that defines

an activity with continuations, you can measure latency between different parts of the system to help track the most expensive stages within the workflow process.

You can use Orchestration Debugger in HAT to query a suspended instance, resume the instance in debug mode, and add appropriate breakpoints using the Technical Details View. This will enable you to trace the activities and debug messages step by step.

You can set breakpoints to track the following events:

- The start and finish of an orchestration
- The start and finish of a shape
- The sending or receipt of a message
- Exceptions

At each breakpoint, you can examine information about local variables, messages and their properties, ports, and role links.

See Also

Finding and Eliminating Bottlenecks

Bottlenecks in the Database Tier

This section explains how to identify bottlenecks in the BizTalk MessageBox database, Tracking database, and BAM Primary Import database. It also explains how to avoid disk contention.

In This Section

- How to Identify Bottlenecks in the MessageBox Database
- How to Identify Bottlenecks in the Tracking Database
- How to Identify Bottlenecks in the BAM Primary Import Database
- How to Avoid Disk Contention

How to Identify Bottlenecks in the MessageBox Database

To identify bottlenecks in the MessageBox database, first ensure that the SQL-Server-Agent Service is started. Change the Service startup state from Manual to Auto so that if the server is restarted, the service will automatically restart.

By default, the BizTalk service will throttle if the Spool, TrackingData, or ApplicationQ tables grow. These tables are pruned by SQL-Agent jobs, which, if not running will cause the Spool to grow causing throttling to start and protect the database from additional pressure. Check the status of the following performance counters:

- BizTalk:Message Agent (Host Name) Message Delivery Throttling State
- BizTalk:Message Agent (Host Name) Message Publishing Throttling State

A value of “0” indicates no throttling is occurring. A value of “6” indicates the system is throttling due to database growth. For information about how to interpret the values of these counters, see What is Host Throttling? (http://go.microsoft.com/fwlink/?LinkID=154694) and Host Throttling

Performance Counters (http://go.microsoft.com/fwlink/?LinkID=155285) in the BizTalk Server 2010 documentation.

Spool table growth

The Spool can start growing for multiple reasons. One reason for Spool growth is if the Application Queues are growing. They could grow due to reasons such as downstream bottlenecks and/or resource contention.

If the Application Queues are small and the Spool is still large, verify that the purge jobs are keeping up. Ensure that the SQL-Agent Service is running and then verify that the following jobs are successfully completing:

- MessageBox_Message_Cleanup_BizTalkMessageBoxDb
- MessageBox_Parts_Cleanup_BizTalkMessageBoxDb

If the MessageZeroSum table is large, it indicates that the messages have been processed. This means that DeQueue has successfully completed and deleted data from the Application Queue tables and the rows have been flagged for deletion. However, the purge jobs are unable to keep up with deleting the data. This can happen if the computer running SQL Server is experiencing severe CPU contention, affecting the ability of the purge jobs to keep up due to CPU starvation.

Application queue table growth

Application Queues host in-flight transition data that, once processed, are cleaned up by DeQueue.

After these messages are processed, the Spool table (holding references to these rows) can be cleaned.

For example, the RxHostQ publishes data to the orchestration PxHostQ. This queue publishes data to the sending TxHostQ each referencing a row in the Spool table. After the messages for a particular HostQ have successfully processed through the system, these rows are deleted by DeQueue. After these rows are deleted, the Spool (which is no longer referenced by these rows) can then be cleaned by the purge jobs.

Application Queue growth indicates that the host instances responsible for draining the Application Queue are unable to keep up with the incoming rate.

For example, the orchestration application queue (PxHostQ) may grow because the server responsible for processing orchestrations is CPU bound and unable to process any faster. However, if the receiving server is fast it may publish faster than what the orchestration server can process leading to the Application Queue growth.

Another reason for orchestration queue growth could be due to memory contention. When many long running orchestration instances are concurrently instantiated in memory, memory bloat indirectly causes throttling to shrink the thread pool until memory pressure is relieved.

A reason why the send application queue may grow is if the downstream system is unable to receive messages (outgoing from BizTalk Server) fast enough. Thus the messages will continue to reside within the BizTalk system causing Application Queue growth. This can cause throttling to start and reduce the receiving rate impacting the overall throughput of the system.

TrackingData table growth

The TrackingData table in the MessageBox database is a transition table into which interceptors write tracking data for both Health and Activity (HAT) and Business Activity Monitoring (BAM) tracking. If tracking is disabled, this table should be empty. By default, HAT-tracking is enabled for the pipelines and orchestrations In/Out events.

If Message Body Tracking is enabled, ensure that the MessageBox database server (that is, the host with "allow host tracking") is running. Ensuring that the host with "allow host tracking" is running will reduce the chance of a bottleneck occurring as the host moves data from TrackingData table in the MessageBox database to the BizTalk Tracking database tables.

It is possible to track custom events by enabling custom HAT-tracking, for example, on promoted properties and message body tracking. In addition to HAT-tracking data, BAM data is also written to the TrackingData table. The Tracking Data Decode Service (TDDS, which runs on the host instance on which tracking is enabled) is responsible for moving this data from the MessageBox database to the BizTalk Tracking and BAM Primary Import databases. Then after the data is successfully moved, TDDS deletes this data. Message body tracking data is moved separately by a SQL-Agent job TrackedMessages_Copy_BizTalkMsgBoxDb.

If TDDS is unable to keep up with the rate at which the interceptors are writing data to the TrackingData table, this table will grow, causing throttling to start. This impacts sustainable throughput. To lessen this problem, ensure at least one host is running with tracking enabled.

If the data still builds up, ensure the BizTalk Tracking database is not growing out of control. Also, ensure the archiving and purging job is running and is able to keep up with the rate at which data is arriving.

Note

By default, the purge job is unable to delete data from the BizTalk Tracking database tables until this data has been archived. If you do not need to archive the tracking data, you can modify the job to purge the BizTalk Tracking database without archiving by following the steps at How to Purge Data from the BizTalk Tracking Database (http://go.microsoft.com/fwlink/?LinkID=153817).

See Also

Bottlenecks in the Database Tier

How to Identify Bottlenecks in the Tracking Database

To identify bottlenecks in the BizTalk Tracking (BizTalkDTADb) database, perform the following steps:

1. Ensure that the SQL-Agent Service is running.
2. Ensure that the Archive/Purge Job is running and completing successfully.
3. Ensure that the TrackedMessages_Copy_BizTalkMsgBoxDB Job is running and completing successfully.
4. Verify that sufficient free disk space is available for the DTADb archives and database growth.

5. Use a dedicated host for tracking and measure the Host Queue Length performance counter when under load.
6. Check the Spool Table size performance counter for an increasing trend over time.
7. Check the Archive/Purge job execution duration for long execution times.
8. Check disk responsiveness (Disk Seconds per Read/Write performance counter) on the disk hosting the BizTalk Tracking database.

We strongly recommend tuning down the value of the following parameters of the dtasp_BackupAndPurgeTrackingDatabase or dtasp_PurgeTrackingDatabase invoked by the DTA Purge and Archive job:

- @nLiveHours tinyint — Any completed instance older than the (live hours) + (live days) will be deleted along with all associated data. Default is 0 hours.
- @nLiveDays tinyint — Any completed instance older than the (live hours) + (live days) will be deleted along with all associated data. Default interval is 1 day.
- @nHardDeleteDays tinyint — All data (even if incomplete) older than this will be deleted. The time interval specified for HardDeleteDays should be greater than the live window of data. The live window of data is the interval of time for which you want to maintain tracking data in the BizTalk Tracking (BizTalkDTADb) database. Anything older than this interval is eligible to be archived at the next archive and then purged. Default is 30 days.

These parameters should be set in accordance with data retention policies in a production environment, whereas in a performance lab tests we recommend that you use values as follows:

```
declare @dtLastBackup datetime set @dtLastBackup = GetUTCDate()
  exec dtasp_PurgeTrackingDatabase 1, 0, 1, @dtLastBackup
```

See Also

Bottlenecks in the Database Tier

How to Identify Bottlenecks in the BAM Primary Import Database

To identify bottlenecks in the Business Activity Monitoring (BAM) database, perform the following steps:

1. Ensure that the Active Instances count is not climbing.
2. Ensure that the SQL-Agent Service is running.
3. If OLAP Analysis is configured, ensure that the BAM_AN_<activityname> job is running at periodic intervals.
4. Ensure that BAM_DM_<activityname> (Data Maintenance) job is scheduled to run at periodic intervals.

   Note
   In high usage scenarios BAM database activity can impact the performance of other BizTalk Server databases, which will affect overall BizTalk Server performance. In this case consider taking the following actions:

   - Consider decreasing the duration of all BAM activities from the default value (6 months) to 1 month or less. This will reduce the time period for which BAM data is maintained in

the BAMPrimaryImport database before being archived. Use the BAM Management Utility `set-activitywindow` command to modify the duration of BAM activities. For more information about the BAM Management Utility activity management commands see Activity Management Commands (http://go.microsoft.com/fwlink/?LinkId=210417).

- Move the BAM Archive database to an instance of SQL Server that does not host any BizTalk MessageBox databases. This will prevent these databases from competing for resources and improve overall performance.

5. Use a dedicated host for tracking and measure the Host Queue Length performance counter when under load.
6. Check the Spool Table size performance counter for an increasing trend over time.
7. Check the Archive/Purge job execution duration for long execution times.
8. Check disk responsiveness (Disk Seconds per Read/Write performance counter) on the disk hosting the BizTalk Tracking database.

See Also

Bottlenecks in the Database Tier

How to Avoid Disk Contention

BizTalk Server is designed as a persistent system. For high throughput scenarios, the MessageBox and BizTalk Tracking databases can experience severe contention. This contention can be aggravated by slow disks. If the disks are slow (greater than 15ms on average for Avg. Disk sec/Read or Avg. Disk sec/Write), it may cause SQL Server to hold onto locks longer (high Lock Wait Time and high Lock Timeouts). This, in turn, can cause the MessageBox tables (Spool and Application Queues) to grow, causing database bloat and throttling. This situation ultimately results in lower overall sustainable throughput.

Note

For information about identifying if a server has a disk bottleneck, see Windows Performance Monitor (http://go.microsoft.com/fwlink/?LinkID=204007). Windows Performance Monitor is a Microsoft Management Console (MMC) snap-in that provides tools for analyzing system performance.

To avoid disk contention, do the following:

Steps	Reference
Use Raid10/0+1 disk configurations.	Best Practices for Avoiding Bottlenecks
If possible, deploy the databases on a high-speed SAN. If multiple databases are sharing the same disks, we recommend configuring them on separate **dedicated** disks. In addition, we recommend separating the MDF and LDF files for the MessageBox database onto	Optimizing Filegroups for the Databases

Steps	Reference
separate disks.	
Consider allocating multiple files for the TEMPDB database, as this will significantly reduce disk contention and spread the load across multiple data files.	Pre-Configuration Database Optimizations
Consider separating the MessageBox database onto a dedicated server that is separate from the BizTalk Tracking databases.	Post-Configuration Database Optimizations
Assign the MSDTC log file directory to a separate dedicated drive.	Optimizing Operating System Performance
If there is contention on the local drive due to the PageFile or MSDTC log, try moving the PageFile and/or the MSDTC log to a separate drive.	Best Practices for Avoiding Bottlenecks
Optimize the Tracking database for write operations.	How to Identify Bottlenecks in the Tracking Database
Optimize the MessageBox database for read and write operations.	How to Identify Bottlenecks in the MessageBox Database
If a BizTalk host instance is saturating the CPU, consider separating sending, receiving, processing, and tracking functionality into multiple hosts. This configures the system so that the orchestration functionality runs on a separate dedicated server to improve overall system throughput.	Optimizing BizTalk Server Performance
If multiple orchestrations are deployed, consider enlisting them in different dedicated orchestration hosts. This isolates the different orchestrations and prevents contention for shared resources either in the same physical address space or on the same server.	Optimizing BizTalk Server Performance
Consider using Windows Performance Monitor to diagnose disk contention issues..	Windows Performance Monitor

For more information about disk performance analysis, see the following resources:

- Ruling Out Disk-Bound Problems (http://go.microsoft.com/fwlink/?LinkId=120947).

- SQL Server Predeployment I/O Best Practices (http://go.microsoft.com/fwlink/?LinkId=120948).
- "I/O Bottlenecks" section of Troubleshooting Performance Problems in SQL Server 2008 (http://go.microsoft.com/fwlink/?LinkID=153586).

See Also

Bottlenecks in the Database Tier

Performance Tools

This topic provides information on tools you can use to evaluate the performance of a BizTalk Server solution. The tools described in this topic have different purposes; some are designed to evaluate end-to-end performance while others focus on evaluating performance of a particular aspect of a BizTalk Server solution.

BizTalk Server Orchestration Profiler

The BizTalk Server Orchestration Profiler is used to view orchestration tracking data for a specified period of time and is useful for determining where performance bottlenecks exist in orchestrations.

For more information about the BizTalk Server Orchestration Profiler, see BizTalk Server 2006 Orchestration Profiler (http://go.microsoft.com/fwlink/?LinkId=102209).

Note

Use of this tool is not supported by Microsoft, and Microsoft makes no guarantees about the suitability of this programs. Use of this program is entirely at your own risk. Also note that this utility was designed to work with BizTalk Server 2006 and therefore may not work as expected on BizTalk Server 2010.

BizUnit 3.0 and BizUnit Designer

BizUnit is a framework designed for automated testing of BizTalk solutions. BizUnit is an excellent tool for testing end-to-end BizTalk Server scenarios. Performing automated testing of BizTalk solutions with BizUnit is the primary focus of the Implementing Automated Testing section of this guide. For more information about BizUnit 3.0, see BizUnit - Framework for Automated Testing of Distributed Systems (http://go.microsoft.com/fwlink/?LinkID=85168).

BizUnit Designer is a GUI that allows for rapid creation of BizUnit test cases that can be used for unit testing or system testing distributed applications. The tool is available at BizUnit Designer (http://go.microsoft.com/fwlink/?LinkID=117917).

Important

As of the writing of this guide, the test cases generated by BizUnit Designer 1.x are only compatible with BizUnit 2.x.

Note

Use of this tool is not supported by Microsoft, and Microsoft makes no guarantees about the suitability of this programs. Use of this program is entirely at your own risk.

IOMeter

IOMeter is an open source tool used for measuring disk I/O performance. For more information about IOMeter, see http://www.iometer.org (http://go.microsoft.com/fwlink/?LinkId=122412).

Note

Use of this tool is not supported by Microsoft, and Microsoft makes no guarantees about the suitability of this programs. Use of this program is entirely at your own risk.

Log Parser

Log parser is a powerful, versatile tool that provides universal query access to text-based data such as log files, XML files and CSV files, as well as key data sources on the Windows® operating system such as the Event Log, the Registry, the file system, and Active Directory®. Log Parser is available for download at Log Parser 2.2 (http://go.microsoft.com/fwlink/?LinkID=100882).

Microsoft BizTalk LoadGen 2007

BizTalk LoadGen 2007 is a load generation tool used to run performance and stress tests against BizTalk Server 2010.

For more information about the Microsoft BizTalk LoadGen 2007 tool, see Microsoft BizTalk LoadGen 2007 Tools (http://go.microsoft.com/fwlink/?LinkId=59841).

Pathping

Pathping provides information about possible data loss at one or more router hops on the way to a target host. To do so, pathping sends Internet Control Message Protocol (ICMP) packets to each router in the path. Pathping.exe is available with all versions of Windows since Windows 2000 Server.

Performance Analysis of Logs (PAL)

The PAL tool is used to generate an HTML-based report that graphically charts important performance counters and generates alerts when thresholds for these counters are exceeded. PAL is an excellent tool for identifying bottlenecks in a BizTalk Server solution to facilitate the appropriate allocation of resources when optimizing the performance of the solution.

For more information about the Performance Analysis of Logs (PAL) tool, see Performance Analysis of Logs (PAL) Tool (http://go.microsoft.com/fwlink/?LinkID=98098).

Note

Use of this tool is not supported by Microsoft, and Microsoft makes no guarantees about the suitability of this programs. Use of this program is entirely at your own risk.

Performance Monitor

Performance Monitor provides a visual display of built-in Windows performance counters, either in real time or as a way to review historical data.

Relog

The Relog utility is used to extract performance counters from logs created by Performance Monitor and convert the data into other formats, such as tab-delimited text files (text-TSV), comma-delimited text files (text-CSV), binary files, and SQL databases. This data can then be analyzed and queried using other tools, such as Log Parser, to generate statistics for key performance indicators (KPIs). The Relog utility is provided with Windows Server 2003 and subsequent versions.

Visual Studio 2010 - Testing the Application

Both Microsoft Visual Studio 2010 Ultimate and Visual Studio Test Professional 2010 now include a new application called Microsoft Test Manager to help you define and manage your testing effort by using test plans. For more information about working with load tests, see Testing the Application (http://go.microsoft.com/fwlink/?LinkID=205342).

Visual Studio 2010 Profiling Tools

The Visual Studio Profiling Tools allows you to profile custom .NET components (custom pipeline components, helper components invoked by pipelines and\or orchestrations, custom functoids). For more information about Visual Studio Profiling Tools see, Analyzing Application Performance by Using Profiling Tools (http://go.microsoft.com/fwlink/?LinkID=210555).

Windows Performance Analysis Tools

Windows Performance Tools are designed for analysis of a wide range of performance problems including application start times, boot issues, deferred procedure calls and interrupt activity (DPCs and ISRs), system responsiveness issues, application resource usage, and interrupt storms.

For more information about the Windows Performance Analysis Tools, see Windows Performance Analysis (http://go.microsoft.com/fwlink/?LinkId=139763).

SQL Server Tools for Performance Monitoring and Tuning

SQL Server provides several tools for monitoring events in SQL Server and for tuning the physical database design. These tools are described in the SQL Server Books Online topic Tools for Performance Monitoring and Tuning (http://go.microsoft.com/fwlink/?LinkId=146357). Information about specific tools used for SQL Server performance monitoring and tuning is provided below:

SQL Profiler

Microsoft SQL Server Profiler can be used to capture Transact-SQL statements that are sent to SQL Server and the SQL Server result sets from these statements. Because SQL Server is tightly integrated with SQL Server, the analysis of a SQL Server Profile trace can be a useful tool for analyzing problems that may occur in BizTalk Server when reading from and writing to SQL Server databases. For information about how to use SQL Server Profiler, see the SQL Server 2008 R2 Books Online topic Using SQL Server Profiler (http://go.microsoft.com/fwlink/?linkid=104423).

Important

There is considerable overhead associated with running SQL Profiler. Therefore SQL Profiler is best suited for use in test or development environments. If using SQL Profiler to troubleshoot a production environment, be aware of the associated overhead costs and limit the use of SQL Profiler accordingly.

Note

When using SQL Profiler to capture Transact-SQL statements, configure SQL Profiler to generate output to a local drive rather than a drive located on a remote network share or other slow device, for example, a local USB memory stick.

SQL Trace

SQL Server provides Transact-SQL system stored procedures to create traces on an instance of the SQL Server Database Engine. These system stored procedures can be used from within your own applications to create traces manually, instead of using SQL Server Profiler. This allows you to write custom applications specific to the needs of your enterprise. For more information about using SQL Trace, see the SQL Server 2008 R2 Books Online topic Introducing SQL Trace (http://go.microsoft.com/fwlink/?LinkId=146354).

Note

When using SQL Trace to capture Transact-SQL statements, configure SQL Trace to generate output to a local drive rather than a drive located on a remote network share or other slow device, such as a USB flash drive.

SQL Activity Monitor

SQL Server 2008 R2 Activity Monitor provides information about SQL Server processes and how these processes affect the current instance of SQL Server. For more information about SQL Server 2008 R2 Activity Monitor, see the SQL Server 2008 R2 Books Online topic Activity Monitor (http://go.microsoft.com/fwlink/?LinkId=146355). For information about how to open Activity Monitor from SQL Server Management Studio, see the SQL Server 2008 R2 Books Online topic How to: Open Activity Monitor (SQL Server Management Studio) (http://go.microsoft.com/fwlink/?LinkId=135094).

SQL Server 2008 R2 Data Collection

SQL Server 2008 R2 provides a data collector that you can use to obtain and save data that is gathered from several sources. The data collector enables you to use data collection containers, which enable you to determine the scope and frequency of data collection on a computer that is running SQL Server 2008 R2. For more information about implementing SQL Server 2008 R2 data collection, see the SQL Server 2008 R2 Books Online topic Data Collection (http://go.microsoft.com/fwlink/?LinkId=146356).

SQLIO

The SQLIO tool was developed by Microsoft to evaluate the I/O capacity of a given configuration. As the name of the tool implies, SQLIO is a valuable tool for measuring the impact of file system I/O on SQL Server performance. SQLIO can be downloaded from SQLIO Disk Subsystem Benchmark Tool (http://go.microsoft.com/fwlink/?LinkId=115176).

See Also

Finding and Eliminating Bottlenecks

Optimizing Performance

A default installation of the Windows operating system, SQL Server, BizTalk Server, and IIS can be significantly optimized to provide optimal performance for a production BizTalk Server environment. WCF configuration parameters can also be tuned from the default settings to provide significantly improved performance. This section provides specific performance optimizations that should be followed when deploying a production BizTalk Server solution.

In This Section

Optimizing Operating System Performance

This section provides recommendations for optimizing performance of the BizTalk Server computers used in a production BizTalk Server environment. The optimizations listed in this section are applied after BizTalk Server has been installed and configured.

Important

Some of the recommendations in this section require modifications to the Windows Server registry. Incorrect use of Registry Editor may cause problems requiring you to reinstall your operating system. Use Registry Editor at your own risk. For more information about how to back up, restore, and modify the registry, see the Microsoft Knowledge Base article Windows registry information for advanced users (http://go.microsoft.com/fwlink/?LinkId=62729).

In This Section

- General Guidelines for Improving Operating System Performance
- **Registry Settings that can be Modified to Improve Operating System Performance**

See Also

Optimizing Performance

General Guidelines for Improving Operating System Performance

The following general guidelines should be followed to improve operating system performance:

Install the latest BIOS, storage area network (SAN) drivers, network adapter firmware and network adapter drivers

Hardware manufacturers regularly release BIOS, firmware, and driver updates that can improve performance and availability for the associated hardware. Visit the hardware manufacturer's Web site to download and apply updates for the following hardware components on each computer in the BizTalk Server environment:

1. BIOS updates
2. SAN drivers (if using a SAN)
3. NIC firmware
4. NIC drivers

Evaluate the usage of Intel Hyper-Threading on BizTalk Server and SQL Server computers

- **Pre-Nehalem Hyper-Threading**:
  - It is critical that hyper-threading be turned off on BizTalk Server computers. This is a BIOS setting, typically found in the Processor settings of the BIOS setup. Hyper-threading makes the server appear to have more processors/processor cores than it actually does; however, hyper-threaded processors typically provide between 20-30% of the performance of a physical processor/processor core. When BizTalk Server counts the number of processors to adjust its self-tuning algorithms, the hyper-threaded processors cause these adjustments to be skewed, which is detrimental to overall performance.
  - Hyper-threading should be turned off on SQL Server computers because applications that can cause high levels of contention (such as BizTalk Server) may cause decreased performance in a hyper-threaded environment on a SQL Server computer.

- **Nehalem Hyper-Threading**: Unlike in older architectures, enabling hyper-threading in Intel microarchitecture “Nehalem” processors can provide up to an almost linear capacity increase. For the best performance results, when you deploy “Nehalem” processors, we recommend that you configure the computer’s BIOS by enabling Intel Hyper-Threading (H-T) Technology for a marked increase in throughput.

Assign the MSDTC log file directory to a separate dedicated drive

In a BizTalk Server environment with multiple MessageBox databases on separate SQL Server computers, additional overhead associated with Microsoft Distributed Transaction Coordinator (MSDTC) is incurred. By default the MSDTC log files are located in the %systemdrive%\windows\system32\msdtc directory of the computers running the DTC service. To mitigate the possibility that DTC logging could become a performance bottleneck, consider moving the MSDTC log file directory to a fast disk drive.

To change the MSDTC log file directory, see Configure DTC Logging (http://go.microsoft.com/fwlink/?LinkID=204107).

Configure antivirus software to avoid real-time scanning of BizTalk Server executables and file drops

Antivirus software real-time scanning of BizTalk Server executable files and any folders or file shares monitored by BizTalk Server receive locations can negatively affect BizTalk Server performance. If antivirus software is installed on the BizTalk Server computer, disable real-time scanning of non-executable file types referenced by any BizTalk Server receive locations (usually .XML, but can also be .csv, .txt, etc.) and configure antivirus software to exclude scanning of BizTalk Server executable files

Disable intrusion detection network scanning between computers in the BizTalk Server environment

Intrusion detection software can slow down or even prevent valid communications over the network. If intrusion detection software is installed, disable network scanning between BizTalk Server computers and external data repositories (SQL Server) computers or messaging services (such as Message Queuing and WebSphere MQSeries) computers.

Defragment all disks in the BizTalk Server environment on a regular basis

Excessive disk fragmentation in the BizTalk Server environment will negatively affect performance. Follow these steps to defragment disks in the BizTalk Server environment:

1. Defragment all disks (local and SAN/NAS) on a regular basis by scheduling off-hours disk defragmentation.
2. Defragment the Windows PageFile and pre-allocate the Master File Tables of each disk in the BizTalk Server environment to boost overall system performance.

> **Note**
> To pre-allocate the Master File Tables, see Knowledge Base article 961095, “About the Master File Table zone reservation in Windows Vista and Windows Server 2008” (http://go.microsoft.com/fwlink/?LinkID=204563).

If antivirus software is installed on the SQL Server computer, disable real-time scanning of data and transaction files

Real-time scanning of the SQL Server data and transaction files (.mdf, .ndf, .ldf, .mdb) can increase disk I/O contention and reduce SQL Server performance. Note that the names of the SQL Server data and transaction files may vary between BizTalk Server environments. For more information about the data and transaction files created with a default BizTalk Server configuration, see Optimizing Filegroups for the Databases.

Configure MSDTC for BizTalk Server and SQL Server

To facilitate transactions between SQL Server and BizTalk Server, you must enable Microsoft Distributed Transaction Coordinator (MSDTC).

To configure Distributed Transaction Coordinator (DTC)

1. Click **Start**, click **Run**, type **dcomcnfg**, and then click **OK** to open **Component Services**.
2. In the console tree, expand **Component Services**, expand **Computers**, expand **My Computer**, expand **Distributed Transaction Coordinator**, and then click **Local DTC**.
3. Right-click **Local DTC**, and then click **Properties** to display the **Local DTC Properties** dialog box.
4. On the **Tracing** tab, under **Output Options** section, clear the **Trace Output** box.
5. Click the **Security** tab.
6. Ensure that each of the following four options is selected, and all others are cleared:
   - **Network DTC Access**
   - **Allow Inbound**
   - **Allow Outbound**
   - **No Authentication Required**
7. Click **OK** to close the **Local DTC Properties** dialog box. If prompted to restart the MSDTC service, click **Yes**.
8. Close **Component Services**.
9. Click **Start**, point to **Administrative Tools**, and then click **Windows Firewall with Advanced Security**.
10. In Windows Firewall with Advanced Security, click **Inbound Rules**.
11. In the **Inbound Rules** pane, right-click **Distributed Transaction Coordinator** * (as appropriate), and then click **Enable Rule**.
12. In Windows Firewall with Advanced Security, click **Outbound Rules**.
13. In the **Outbound Rules** pane, right-click **Distributed Transaction Coordinator** * (as appropriate), and then click **Enable Rule**.
14. On the **Control Panel**, double-click **Administrative Tools**.
15. In the right-hand pane, double-click **Services**.
16. In the right-hand pane of **Services (Local)**, right-click **COM+ System Application**, click **Restart**, and wait for the service to restart.
17. Right-click and restart the **Distributed Transaction Coordinator** service.

18. Right-click and restart the **SQL Server (MSSQLSERVER)** service.
19. Close **Services (Local)**, and then close **Administrative Tools**.

Configure firewall(s) for BizTalk Server

Note

This step is only required if one or more firewalls are in place in your BizTalk Server environment.

Review the following information to configure firewall(s) for BizTalk Server:

- Required Ports for BizTalk Server (http://go.microsoft.com/fwlink/?LinkID=153238).
- To configure RPC dynamic port allocation to work with firewalls, see Knowledge Base article 929851, "The default dynamic port range for TCP/IP has changed in Windows Vista and in Windows Server 2008" (http://go.microsoft.com/fwlink/?LinkID=204568). For information about how to configure Windows Firewall to accommodate the necessary ports, see Windows Firewall and IPsec Policy Deployment Step-by-Step Guide (http://go.microsoft.com/fwlink/?LinkID=204569).

Install appropriate COM+ and MSDTC hotfix rollup packages

Review the following information to install the appropriate COM+ and MS DTC hotfix rollup packages:

- The MS DTC hotfix can be found in Microsoft Knowledge Base article978476 "The MS DTC issue that is fixed in Windows Server 2008 R2 MS DTC Hotfix Rollup Package 1" (http://go.microsoft.com/fwlink/?LinkID=204109).
- The latest DTC hotfix rollup package KB article can be found by searching http://support.microsoft.com (http://go.microsoft.com/fwlink/?LinkID=96185) for the phrase (including the quotes):

  ```
  "MS DTC Hotfix Rollup Package"
  ```

  The following query does this search for you. Choose the latest one: http://support.microsoft.com/search/default.aspx?query="MS+DTC+Hotfix+Rollup+Package"

Use the Interrupt-Affinity Policy Tool to bind network adapter interrupts to specific processors on multiprocessor computers

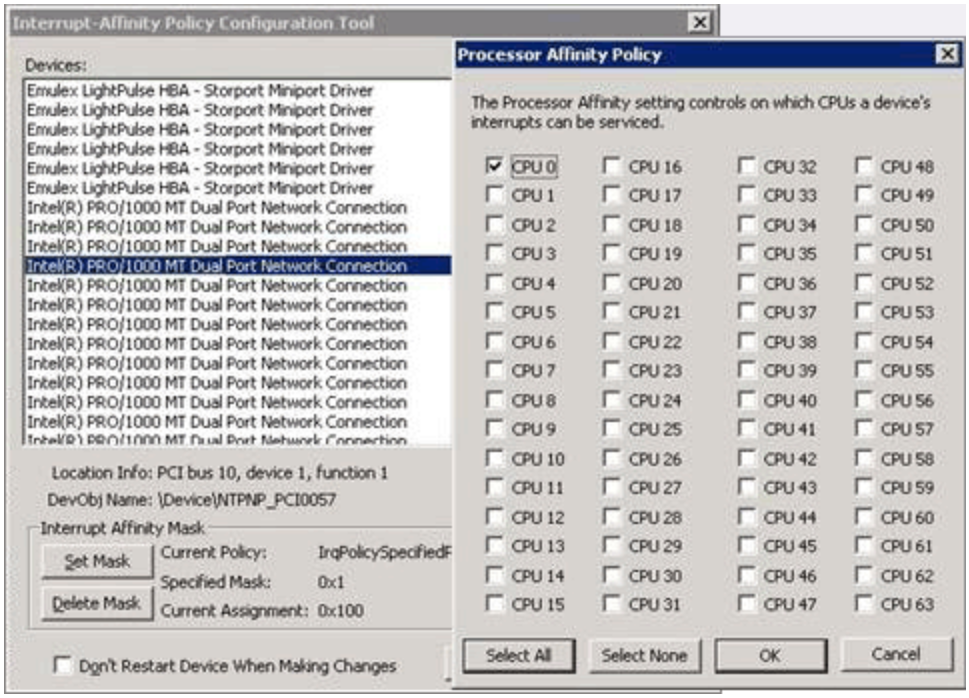
The Interrupt-Affinity Policy (IntPolicy) is a tool that allows you to “bind” or change the CPU affinity of the interrupts for a given device (such as a network adapter) to a specific processor or processors on a multiprocessor computer. This binding is also referred to as partitioning. The binding of interrupts from a specific network adapter to specific processors on a multiprocessor computer enforces running deferred procedure calls (DPCs) and interrupt service routines (ISRs) for the network adapter on the designated processors. Note that interrupt affinity cannot be configured on single processor computers.

Note

A DPC is defined as a queued call to a kernel-mode function that will usually be executed at a later time. An ISR is defined as a routine whose purpose is to service a device when it generates an interrupt. For more information about deferred procedure calls and

interrupt service routines, see the Windows Driver Kit documentation (http://go.microsoft.com/fwlink/?LinkId=84418).

Interrupt-Affinity Policy Tool



On Windows Server 2008 based multiprocessor computers, the default behavior of the interrupt controller is to assign device interrupts to any available processor. When network connections and file server sessions for a given network adapter are bound/partitioned to run on a specific set of processors, rather than any available processor, the performance and scalability of the associated network processing is improved. Large BizTalk Server solutions often employ the use of multi-processor SQL Server computers with multiple network adapters for which interrupt-binding may be particularly beneficial.

Interrupt-binding using IntPolicy should always be evaluated in a test environment before employing in a production environment. The hardware, operating system and application configuration of the test environment should approximate the production environment as closely as possible. This will allow you to test various permutations of interrupt-binding and determine the extent that interrupt-binding will increase performance.

We recommend that you disable hyper-threading before configuring IntPolicy on a computer with CPUs that supports hyper-threading. This will ensure that interrupts are assigned to physical processors rather than logical processors. Assigning interrupt affinity to logical processors that refer to the same physical processor will not increase performance and could even degrade system performance. The Interrupt-Affinity Policy Tool (http://go.microsoft.com/fwlink/?LinkID=204111) is available for download from the WHDC website.

Use the NTFS file system on all volumes

Windows Server offers multiple file system types for formatting drives, including NTFS, FAT, and FAT32. NTFS should always be the file system of choice for servers.

NTFS offers considerable performance benefits over the FAT and FAT32 file systems and should be used exclusively on Windows servers. In addition, NTFS offers many security, scalability, stability and recoverability benefits over FAT and FAT32.

Under previous versions of Windows, FAT and FAT32 were often implemented for smaller volumes (say <500 MB) because they were often faster in such situations. With disk storage relatively inexpensive today and operating systems and applications pushing drive capacity to a maximum, it is unlikely that such small volumes will be in use. FAT32 scales better than FAT on larger volumes but is still not an appropriate file system for Windows servers.

FAT and FAT32 have often been implemented in the past as they were seen as more easily recoverable and manageable with native DOS tools in the event of a problem with a volume. Today, with the various NTFS recoverability tools built both natively into the operating system and available as third-party utilities available, there should no longer be a valid argument for not using NTFS for file systems.

Do not use NTFS file compression

Though using NTFS file system compression is an easy way to reduce space on volumes, it is not appropriate for enterprise file servers. Implementing compression places an unnecessary overhead on the CPU for all disk operations and is best avoided. Think about options for adding additional disks, near-line storage or consider archiving data before seriously considering file system compression.

Review disk controller stripe size and volume allocation units

When configuring drive arrays and logical drives within your hardware drive controller, ensure you match the controller stripe size with the allocation unit size that the volumes will be formatted with. This will ensure disk read and write performance is optimal and offer better overall server performance.

Configuring larger allocation unit (or cluster or block) sizes will cause disk space to be used less efficiently, but will also provide higher disk I/O performance as the disk head can read in more data during each read activity.

To determine the optimal setting to configure the controller and format the disks with, you should determine the average disk transfer size on the disk subsystem of a server with similar file system characteristics. Use the Windows Performance Monitor tool to monitor the Logical Disk object counters of Avg. Disk Bytes/Read and Avg. Disk Bytes/Write over a period of normal activity to help determine the best value to use.

Although smaller allocation unit sizes may be warranted if the system will be accessing many small files or records, an allocation unit size of 64 KB delivers sound performance and I/O throughput under most circumstances. Improvements in performance with tuned allocation unit sizes can be particularly noted when disk load increases.

Note

Either the FORMAT command line tool or the Disk Management tool is required to specify an allocation unit size larger than 4096 bytes (4 KB) when formatting volumes. Windows Explorer will only format up to this threshold. The CHKDSK command can be used to confirm the current allocation unit size of a volume however it needs to scan the entire volume before the desired information is displayed (shown as Bytes in each allocation unit).

Monitor drive space utilization

The less data a disk has on it, the faster it will operate. This is because on a well defragmented drive, data is written as close to the outer edge of the disk as possible, as this is where the disk spins the fastest and yields the best performance.

Disk seek time is normally considerably longer than read or write activities. As noted above, data is initially written to the outside edge of a disk. As demand for disk storage increases and free space reduces, data is written closer to the center of the disk. Disk seek time is increased in locating the data as the head moves away from the edge, and when found, it takes longer to read, hindering disk I/O performance.

This means that monitoring disk space utilization is important not just for capacity reasons but for performance also.

As a rule of thumb, work towards a goal of keeping disk free space between 20% to 25% of total disk space. If free disk space drops below this threshold, then disk I/O performance will be negatively impacted.

Implement a strategy to avoid disk fragmentation

Run a defragmenter utility regularly on your disks, including the root drive, to prevent performance degradation. Do this weekly on busy disks. A disk defragmenter is installed with Windows and can be run from a Scheduled Task at specified intervals.

Optimize Windows Server performance for background services

The BizTalk Server process (BTSNTSVC.exe) runs as a background service.

Windows Server 2008 uses preemptive multi-tasking to prioritize process threads that will be attended to by the CPU. Preemptive multi-tasking is a methodology whereby the execution of a process is halted and another process is started, at the discretion of the operating system. This scheme prevents a single thread from dominating the CPU.

Switching the CPU from executing one process to the next is known as context-switching. The Windows operating system includes a setting that determines how long individual threads are allowed to run on the CPU before a context-switch occurs and the next thread is serviced. This amount of time is referred to as a quantum. This setting lets you choose how processor quanta are shared between foreground programs and background services. Typically for a server it is not desirable to allow a foreground program to have more CPU time allocated to it than background services. That is, all applications and their processes running on the server should be given equal consideration for CPU time.

To increase performance for background service like BizTalk host instances, follow these steps:

1. Click **Start**, click **Control Panel**, and then click **System**.
2. Click the **Advanced** tab, and then click **Settings** under **Performance**.
3. Click the **Advanced** tab, click **Background services**, and then click **OK** twice.

Disable non-essential services

A default installation of Windows Server 2008 enables several services that may not be required in a BizTalk Server environment. Each running service consumes system resources and so unnecessary services should be disabled to improve overall performance.
Care should be taken when disabling services. Thoroughly research the purpose of a service before disabling the service as Windows Server requires certain services are running. If services required by Windows Server 2008 are disabled, the operating system may become inoperable and possibly even unable to boot.
To disable Windows Server 2008 services that are not required for a dedicated BizTalk Server, follow these steps:

1. Click **Start**, point to **Administrative Tools**, and then click **Computer Management**.
2. Under **Computer Management (Local)**, expand **Services and Applications**, and then click **Services**.
   In the **Status** column, each service that is running is labeled "Started." Stop and disable any service that is started unnecessarily, for example, the following services are not required on a dedicated BizTalk Server:
   - Alerter
   - ClipBook
   - DHCP Server
   - Fax Service
   - File Replication
   - Infrared Monitor
   - Internet Connection Sharing
   - Messenger
   - NetMeeting Remote Desktop Sharing
   - Network DDE
   - Network DDE DSDM
   - NWLink NetBIOS
   - NWLink IPX/SP
   - Print Spooler
   - Telephony
   - Telnet
   - Uninterruptible Power Supply
3. Note the services that depend on each service that you want to disable. To do this, follow these steps:
   a. Double-click the service you want to disable.

b. Click the **Dependencies** tab.

c. In the **This service depends on the following system components** list, note the services this service depends on.

d. In the **The following system components depend on this service** list, note the services that cannot start without this service, and then click **OK**.

4. One at a time, disable each service you have selected. To do this, follow these steps:

   a. Right-click the service you want to disable, and then click **Properties**.

   b. In the **Startup type** list, click **Disabled**.

   c. If you want to stop the service immediately, click **Stop**.

      If the **Stop Other Services** dialog box appears, note the other dependent services that will also stop, and then click **Yes**, and then click **OK**.

5. Repeat step 4 to disable the other nonessential services.

Note

Test the server for correct operation after you disable each service to make sure you did not disable a service you want to continue to use.

If the server is a member of a Windows Server 2008 domain, which BizTalk Servers often are, you must have the TCP/IP helper service on your system to correctly apply Group Policy to the computer.

When you disable the DHCP client, the DHCP client stops DNS dynamic update protocol registration and requires manual DNS records to be added for this client to the DNS server.

Manually load Microsoft Certificate Revocation lists

When starting a .NET application, the .NET Framework will attempt to download the Certificate Revocation list (CRL) for any signed assembly. If your system does not have direct access to the Internet, or is restricted from accessing the Microsoft.com domain, this may delay startup of BizTalk Server. To avoid this delay at application startup, you can use the following steps to manually download and install the code signing Certificate Revocation Lists on your system.

1. Download the latest CRL updates from http://crl.microsoft.com/pki/crl/products/CodeSignPCA.crl (http://go.microsoft.com/fwlink/?LinkID=117794) and http://crl.microsoft.com/pki/crl/products/CodeSignPCA2.crl (http://go.microsoft.com/fwlink/?LinkId=117795).
2. Move the CodeSignPCA.crl and CodeSignPCA2.crl files to the isolated system.
3. From a command prompt, enter the following command to use the certutil utility to update the local certificate store with the CRL downloaded in step 1:

   certutil –addstore CA c:\CodeSignPCA.crl

The CRL files are updated regularly, so you should consider setting a reoccurring task of downloading and installing the CRL updates. To view the next update time, double-click the .crl file and view the value of the **Next Update** field.

Synchronize time on all servers

Many operations involving tickets, receipts and logging rely on the local system clock being accurate. This is especially true in a distributed environment, where time discrepancies between systems may cause logs to be out of sync or tickets issued by one system to be rejected by another as expired or not yet valid.

For more information on configuring a server to automatically synchronize time, see Configure a client computer for automatic domain time synchronization (http://go.microsoft.com/fwlink/?LinkId=99420).

Configure the Windows PAGEFILE for optimal performance

Follow these guidelines to configure the Windows PAGEFILE (paging file) for optimal performance:

1. **Move the paging file to a physical volume separate from the physical drive that operating system is installed on to reduce disk contention and increase disk performance** - On BizTalk Server computers, the performance gain associated with moving the paging file will vary depending on the document processing load. On SQL Server computers, moving the paging file to a separate volume is considered a best practice in all scenarios due to the disk intensive nature of SQL Server.
2. **Isolate the paging file onto one or more dedicated physical drives that are configured as either RAID-0 (striping) or RAID-1 (mirroring) arrays, or on single disks without RAID** - By using a dedicated disk or drive array where PAGEFILE.SYS is the only file on the entire volume, the paging file will not become fragmented, which will also improve performance. As with most disk-arrays, performance of the array is improved as the number of physical disks in the array is increased. If the paging file is distributed between multiple volumes on multiple physical drives in a disk array, the paging file size should be the same size on each drive in the array. When configuring a disk array, it is also recommended to use physical drives that have the same capacity and speed. Note that redundancy is not normally required for the paging file.
3. **Do not configure the paging file on a RAID 5 array** - Configuration of the paging file on a RAID 5 array is not recommended because paging file activity is write intensive and RAID 5 arrays are better suited for read performance than write performance.
4. **If you do not have resources to move the paging file to a physical volume other than the operating system is installed on, configure the paging file to reside on the same logical volume as the operating system** - Configuring the paging file to reside on a another logical volume which is on the same physical disk as the operating system will increase disk seek time and reduce system performance as the disk drive platter heads will be continually moving between the volumes, alternately accessing the page file, operating system files, application files, and data files. Also, the operating system is typically installed on the first partition of a physical disk, which is usually the closest to the outside edge of the physical disk and where disk speed is and associated performance are optimal for the disk.

   Important
   If you do remove the paging file from the boot partition, Windows cannot create a crash dump file (MEMORY.DMP) in which to write debugging information in the event that a kernel mode STOP error occurs. If you do require a crash dump file, then you

will have no option but to leave a paging file of at least the size of physical memory + 1 MB on the boot partition.

5. **Manually set the size of the paging file** – Manually setting the size of the paging file typically provides better performance than allowing the server to size it automatically or having no paging file at all. Best-practice tuning is to set the initial (minimum) and maximum size settings for the paging file to the same value. This ensures that no processing resources are lost to the dynamic resizing of the paging file, which can be intensive. This is especially true given that this resizing activity typically occurs when the memory resources on the system are already becoming constrained. Setting the same minimum and maximum page file size values also ensure the paging area on a disk is one single, contiguous area, improving disk seek time. Windows Server 2008 automatically recommends a total paging file size equal to 1.5 times the amount of installed RAM. On servers with adequate disk space, the paging file on all disks combined should be configured up to twice the physical memory for optimal performance.

Remove CPU-intensive screen savers

3D or OpenGL screen savers are known to be CPU-intensive and use important system resources when they are running. It is best to avoid installing these altogether as an option at server-build time, or to remove them if they have been installed. The basic “Windows Server 2008” or blank screen savers are an excellent alternative to using CPU-intensive screen savers.

See Also

Optimizing Operating System Performance

Optimizing Network Performance

In a BizTalk Server environment where the BizTalk Server computer is separate from the SQL Server computer, each and every message processed by BizTalk Server requires communication over the network. This communication includes considerable traffic between the BizTalk Server computer and the BizTalk MessageBox database, the BizTalk Management database, the BAM databases, and other databases. In high-load scenarios, this communication can result in considerable network traffic and can become a bottleneck, especially when network settings have not been optimized, not enough network interface cards are installed, or insufficient network bandwidth is available.

This section provides some general recommendations for improving network performance and describes several registry entries that can be modified to optimize the network stack to mitigate the occurrence of bottlenecks on the network.

Important

Some of the recommendations in this section require modifications to the Windows registry. Incorrect use of Registry Editor may cause problems requiring you to reinstall your operating system. Use Registry Editor at your own risk. For more information about how to back up, restore, and modify the registry, see the Microsoft Knowledge Base article 256896, "Windows registry information for advanced users" (http://go.microsoft.com/fwlink/?LinkId=62729).

In This Section

- General Guidelines for Improving Network Performance
- Registry Settings that can be Modified to Improve Network Performance

See Also

Optimizing Performance

General Guidelines for Improving Network Performance

Adjusting network settings to optimal values has been shown to effectively address network bottlenecks and improve overall network performance in BizTalk Server solutions. This should be done on all computers involved in the solution, including BizTalk Server computers, SQL Server computers, and any other server computers.

Note

The most common indicator that Network IO is a bottleneck in a BizTalk Server environment is the counter “SQL Server:Wait Statistics\Network IO waits”. When the value for **Avg Wait Time** in this counter is greater than zero on one or more of your SQL Server computers, then Network IO is a bottleneck.

The following recommendation can be used to increase network performance:

Add additional network cards to computers in the BizTalk Server environment

Just as adding additional hard drives can improve disk performance, adding additional network cards can improve network performance. If the network cards on the computers in your BizTalk Server environment are saturated and the card is a bottleneck, consider adding one or more additional network cards to improve performance.

Implement network segmentation

Follow the recommendations in the **Subnets** section of the "BizTalk Server Database Optimization" whitepaper (http://go.microsoft.com/fwlink/?LinkID=101578).

Where possible, replace hubs with switches

Switches contain logic to directly route traffic between the source and destination whereas hubs use a broadcast model to route traffic. Therefore switches are more efficient and offer improved performance.

Remove unnecessary network protocols

Windows Server computers sometimes have more network services and protocols installed than are actually required. Each additional network client, service or protocol places additional overhead on system resources.
In addition, each installed protocol generates network traffic. By removing unnecessary network clients, services and protocols, system resources are made available for other processes, excess network traffic is avoided and the number of network bindings that must be negotiated is reduced

to a minimum.
To see the currently installed network clients, protocols and services, follow these steps:

1. Click **Start**, and then click **Control Panel**.
2. In Control Panel, do one of the following
    a. In **Adjust your computer's settings**, set **View by** to **Category**, click **Network and Internet**, and then click **Network and Sharing Center**.
    b. In **Adjust your computer's settings**, set **View by** to either **Large icons** or **Small icons**, and then click **Network and Sharing Center**.
3. In the Tasks pane, click **Change adapter settings**.
4. Right-click **Local Area Connection** (or the entry for your network connection), and then click **Properties** to display the properties dialog box for the network connection.
5. To remove an unnecessary item, select it and click **Uninstall**. To disable an item, simply clear the checkbox associated with the item.

If you are unsure about the effects of uninstalling an item for the connection, then disable the item rather than uninstalling it. Disabling items allows you to determine which services, protocols and clients are actually required on a system. When it has been determined that disabling an item has no adverse affect on the server, the item can then be uninstalled.
In many cases, only the following three components are required for operation on a standard TCP/IP based network:

- Client for Microsoft Networks
- File and Printer Sharing for Microsoft Networks
- Internet Protocol (TCP/IP)

Network adapter drivers on all computers in the BizTalk Server environment should be tuned for performance

Important
Before applying tuning to network adapter drivers always install the latest network adapter device drivers for the network cards in the environment.

Adjust the network adapter device drivers to maximize the amount of memory available for packet buffering, both incoming and outgoing. Also maximize buffer counts, especially transmit buffers and coalesce buffers. The default values for these parameters, and whether they are even provided, vary between manufacturers and driver versions. The goal is to maximize the work done by the network adapter hardware, and to allow the greatest possible buffer space for network operations to mitigate network traffic bursts and associated congestion.

Note
Steps to tune network adapter drivers vary by manufacturer.

Follow these steps to access settings for network adapters:

1. Click **Start** and then click **Control Panel**.
2. In Control Panel, do one of the following:

a. In **Adjust your computer's settings**, set **View by** to **Category**, click **Network and Internet**, and then click **Network and Sharing Center**.
b. In **Adjust your computer's settings**, set **View by** to either **Large icons** or **Small icons**, and then click **Network and Sharing Center**.

3. In the Tasks pane, click **Change adapter settings**.
4. Right-click **Local Area Connection** (or the name of your network connection), and then click **Properties**.
5. On the **Networking** tab, click **Configure**.
6. Click the **Advanced** tab to access properties that can be configured for the network adapter.

The following properties should be configured for each network adapter in the BizTalk Server environment:

Note

You apply these settings for each physical network adapter, including the individual network adapters within a teamed set of network adapters that are configured for aggregation, load balancing, or fault tolerance. With some teaming software, you might need to apply these settings to the team also. Note that some network adapters are self-tuning and may not offer the option to configure parameters manually.

- **Power Option** – Configure the network adapter driver to prevent power management functionality from turning off the network adapter to save power. This functionality may be useful for client computers but should seldom, if ever, be used on a BizTalk Server or SQL Server computer.
- **Fixed Speed/Duplex (do not use AUTO)** - It is very important that the network speed, duplex, and flow control parameters are set to correspond to the settings on the switch to which they are connected. This will mitigate the occurrence of periodic "auto-synchronization" which may temporarily take connections off-line.
- **Max Coalesce Buffers** - Map registers are system resources used to convert physical addresses to virtual addresses for network adapters that support bus mastering. Coalesce buffers are available to the network driver if the driver runs out of map registers. Set this value as high as possible for maximum performance. On servers with limited physical memory, this may have a negative impact as coalesce buffers consume system memory. On most systems however, the maximum setting can be applied without significantly reducing available memory.
- **Max Transmit/Send Descriptors and Send Buffers** - This setting specifies how many transmit control buffers the driver allocates for use by the network interface. This directly reflects the number of outstanding packets the driver can have in its "send" queue. Set this value as high as possible for maximum performance. On servers with limited physical memory, this may have a negative impact as send buffers consume system memory. On most systems however, the maximum setting can be applied without significantly reducing available memory.
- **Max Receive Buffers** - This setting specifies the amount of memory buffer used by the network interface driver when copying data to the protocol memory. It is normally set by default to a relatively low value. Set this value as high as possible for maximum performance. On servers with limited physical memory, this may have a negative impact as receive buffers

consume system memory. On most systems however, the maximum setting can be applied without significantly reducing available memory.

- **All offload options ON** - In almost all cases performance is improved when enabling network interface offload features. Some network adapters provide separate parameters to enable or disable offloading for send and receive traffic. Offloading tasks from the CPU to the network adapter can help lower CPU usage on the server which will improve overall system performance. The Microsoft TCP/IP transport can offload one or more of the following tasks to a network adapter that has the appropriate capabilities:
  - **Checksum tasks** - The TCP/IP transport can offload the calculation and validation of IP and TCP checksums for sends and receives to the network adapter; enable this option if the network adapter driver provides this capability.
  - **IP security tasks** - The TCP/IP transport can offload the calculation and validation of encrypted checksums for authentication headers (AH) and encapsulating security payloads (ESP) to the network adapter. The TCP/IP transport can also offload the encryption and decryption of ESP payloads to the network adapter. Enable these options if the network adapter driver provides this capability.
  - **Segmentation of large TCP packets** - The TCP/IP transport supports large send offload (LSO). With LSO, the TCP/IP transport can offload the segmentation of large TCP packets.
  - **Stack Offload** – The entire network stack can be offloaded to a network adapter that has the appropriate capabilities. Enable this option if the network adapter driver provides this capability.
- **Wake On LAN disabled (unless being used)** – Configure the network adapter driver to disable wake-on lan functionality. This functionality may be useful for client computers but should seldom if ever be used on a BizTalk Server or SQL Server computer.

For more information about tuning network adapters for performance, see the **Network Device Settings** section of the "BizTalk Server Database Optimization" whitepaper (http://go.microsoft.com/fwlink/?LinkID=101578).

See Also

Registry Settings that can be Modified to Improve Network Performance

Registry Settings that can be Modified to Improve Network Performance

This topic provides a description of recommended values for several registry entries that impact network performance. These registry entries can be applied manually or can be applied via the operating system optimization PowerShell script included in Windows PowerShell Scripts.

Important

During performance testing completed for this guide it was observed that Windows Server 2008 appears to be tuned by default. Modification of these registry settings should only be done after a careful analysis of the effects on the system.

Registry settings that can be modified to improve network performance

The **DisablePagingExecutive** value governs whether or not Windows will page the NT executive to disk. Setting this entry to a value of 1 will prevent pageable drivers and system code in the Windows NT Executive from being paged out to disk. Although this decreases the response time in systems with extremely large amounts of physical memory (RAM), it is critical that there is enough RAM installed, otherwise the server could be rendered unstable. For more information about the **DisablePagingExecutive** registry entry, see DisablePagingExecutive (http://go.microsoft.com/fwlink/?LinkId=113707).

DisablePagingExecutive

Key:	HKLM:\System\CurrentControlSet\Control\Session Manager\Memory Management
Value:	DisablePagingExecutive
Data Type:	REG_DWORD
Range:	0 to 1
Default value:	0
Recommended value:	1
Value exists by default?	**Yes**

The **IRPStackSize** value specifies the number of stack locations in I/O request packets (IRPs) that are used by Windows Server 2008 R2. You may have to increase this number for certain transports, for media access control (MAC) drivers, or for file system drivers. Each stack uses 36 bytes of memory for each receive buffer. For more information about the IRPStackSize registry entry, see Microsoft Knowledge Base Article 285089, "Description of the IRPStackSize parameter in Windows 2000, in Windows XP, and in Windows Server 2003" (http://go.microsoft.com/fwlink/?LinkID=105643).

IRPStackSize

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\LanmanServer\ Parameters
Value:	IRPStackSize
Data Type:	REG_DWORD
Range:	11 to 50
Default value:	15
Recommend	32

ed value:	
Value exists by default?	**No**, needs to be added.

The **SizReqBuf** value specifies the size in bytes of the raw receive buffers (work items) that the Server service uses. Small work items use less memory, but large work items can improve performance. The value that works best in a particular environment depends on the configuration of that environment. For an optional value, you might try increasing the value as high as 4410 (hexadecimal); this has been shown to work well in a fairly standard Ethernet environment. However, going over setting a value over 4000 hexadecimal has been seen to cause other issues on some servers. Therefore, the default starting point for the **SizeReqBuf** entry should be 4000 hexadecimal (16384 decimal). By default, the value for this entry is 4356 bytes on servers with less than 512 MB of memory. On servers with more than 512 MB of memory, this value is increased to 16384 bytes (16 KB). A receive buffer that is larger can improve performance on directory queries and similar commands, but at the price of more memory per work item.

Increasing the **SizReqBuf** value can increase performance significantly in a high-latency environment. However, note that increasing the **SizReqBuf** value also increases the non-paged pool memory used by the Server service. If you increase the **SizReqBuf** value, monitor non-paged pool memory to make sure that the change does not adversely impact the performance of the server.

SizReqBuf

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\LanmanServer\Parameters
Value:	SizReqBuf
Data Type:	REG_DWORD
Range:	1-65535
Default value:	16,384 (bytes) on servers with 512 MB or more or physical memory, 4,356 (bytes) on servers with less than 512 MB physical memory.
Recommended value:	17424 (bytes) on servers with 512 MB or more or physical memory, 4,356 (bytes) on servers with less than 512 MB physical memory.
Value exists by default?	**No**, needs to be added.

Review the following information to configure TCP/IP registry settings for optimal performance:

- Avoiding TCP/IP Port Exhaustion (http://go.microsoft.com/fwlink/?LinkID=155309).

- Review the "HKLM\SYSTEM\CurrentControlSet\ Services\Tcpip\Parameters" section of the "BizTalk Server Database Optimization" whitepaper (http://go.microsoft.com/fwlink/?LinkID=101578).

The **DefaultTTL** value specifies the default time-to-live (TTL) value set in the header of outgoing IP packets. The TTL determines the maximum amount of time that an IP packet may live in the network without reaching its destination. It is effectively a limit on the number of links on which an IP packet is allowed to travel before being discarded.

DefaultTTL

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Value:	DefaultTTL
Data Type:	REG_DWORD
Range:	1 to 255 (seconds)
Default value:	128
Recommended value:	64
Value exists by default?	**No**, needs to be added

The **EnablePMTUDiscovery** value governs whether TCP will attempt to discover the Maximum Transmission Unit (MTU), or largest packet size for the entire path to a remote host. By discovering the Path MTU (PMTU) and limiting TCP segments to this size, TCP can eliminate packet fragmentation at routers along the path that connect networks with different MTUs. Fragmentation adversely affects TCP throughput and causes network congestion. Setting this parameter to 0 (or off) causes an MTU of 576 bytes to be used for all connections to destinations other than the local subnet.

Important
This entry should not be set to a value of 1 if the server is directly exposed to potential attackers.

EnablePMTUDiscovery

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Value:	EnablePMTUDiscovery
Data Type:	REG_DWORD

Range:	0 to 1
Default value:	1
Recommended value:	1
Value exists by default?	**No**, needs to be added.

The **EnablePMTUBHDetect** value governs whether TCP tries to detect black hole routers during the Path MTU (maximum transmission unit) discovery process. Enabling black hole detection increases the maximum number of times TCP retransmits a given segment. If the value of this entry is 1, TCP recognizes when it has transmitted the same segment several times without receiving an acknowledgement. It reduces the maximum segment size (MSS) to 536 bytes, and it sets the Don't-Fragment bit. If, as a result, receipt of the segment is acknowledged, TCP continues this practice in all subsequent transmissions on the connection.

EnablePMTUBHDetect

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Value:	EnablePMTUBHDetect
Data Type:	REG_DWORD
Range:	0 to 1
Default value:	0
Recommended value:	1
Value exists by default?	**No**, needs to be added.

The **TcpMaxDupAcks** value determines the number of duplicate ACKs that must be received for the same sequence number of sent data before fast retransmit is triggered to resend the segment that has been dropped in transit. If you set the value of this entry to 1, then the system retransmits a segment when it receives an ACK for a segment with a sequence number that is less than the number of the segment currently being sent.
When data arrives with a sequence number that is greater than expected, the receiver assumes that data with the expected number was dropped, and it immediately sends an ACK with the ACK number set to the expected sequence number. The receiver sends ACKs set to the same missing number each time it receives a TCP segment that has a sequence number greater than

expected. The sender recognizes the duplicate ACKs and sends the missing segment. The recommended value of 2 is also the default value but Windows Server 2008 R2 does not add this entry to the registry, so it should be added.

TcpMaxDupAcks

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Value:	TcpMaxDupAcks
Data Type:	REG_DWORD
Range:	1 to 3
Default value:	2
Recommended value:	2
Value exists by default?	**No**, needs to be added.

The **Tcp1323Opts** value governs whether TCP uses the timestamping and window scaling features described in RFC 1323, TCP Extensions for High Performance. Window scaling permits TCP to negotiate a scaling factor for the TCP receive window size, allowing for a very large TCP receive window of up to 1 GB. The TCP receive window is the amount of data that the sending host can send at one time on a connection. Timestamps help TCP measure round trip time (RTT) accurately in order to adjust retransmission timeouts. The Timestamps option provides two timestamp fields of 4 bytes each in the TCP header, one to record the time the initial transmission is sent and one to record the time on the remote host. This entry is a 2-bit bitmask. The lower bit determines whether scaling is enabled; the higher bit determines whether timestamps are enabled.

Tcp1323Opts

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Value:	Tcp1323Opts
Data Type:	REG_DWORD
Range:	0 to 3
Default value:	3
Recommended value:	1 (also consider setting to a value of 3 if high packet loss / retransmits are occurring).

Value exists by default?	**No**, needs to be added.

The **SackOpts** value governs whether the Selective Acknowledgment (SACK) feature ofWindows Server 2008 R2 TCP/IP is enabled. SACK is an optimizing feature based upon RFC 2018, TCP Selective Acknowledgement Options. SACK permits receipt acknowledgement of individual blocks of data in a continuous sequence, rather than just the last sequence number. When SACK is enabled, the recipient can tell the sender that one or more data blocks are missing from the middle of a sequence, and the sender can retransmit only the missing data.

SackOpts

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Value:	SackOpts
Data Type:	REG_DWORD
Range:	0 to 1
Default value:	1
Recommended value:	1
Value exists by default?	**No**, needs to be added.

The **MaxFreeTcbs** value determines the number of TCP control blocks (TCBs) the system creates to support active connections. Because each connection requires a control block, this value determines how many active connections TCP can support simultaneously. If all control blocks are used and more connection requests arrive, TCP can prematurely release connections in the TIME_WAIT state in order to free a control block for a new connection.

Note
If the value for this entry is increased, then the value for the **MaxHashTableSize** value should also be increased.

MaxFreeTcbs

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Value:	MaxFreeTcbs

Data Type:	REG_DWORD
Range:	0 to 4294967295
Default value:	Varies with the system and amount of physical memory on the computer. For more information, see Appendix A: TCP/IP Configuration Parameters (http://go.microsoft.com/fwlink/?LinkId=113716).
Recommended value:	65535
Value exists by default?	**No**, needs to be added.

The **MaxHashTableSize** value controls how fast the system can find a TCB and should be increased if the MaxFreeTcbs value is increased from its default value.

Note
This value should be set to a power of 2 (for example, 512, 1024, 2048, and so on.) If this value is not a power of 2, the system configures the hash table to the next power of 2 value (for example, a setting of 513 is rounded up to 1024).

MaxHashTableSize

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Value:	MaxHashTableSize
Data Type:	REG_DWORD
Range:	64 to 65536
Default value:	512
Recommended value:	65536
Value exists by default?	**No**, needs to be added.

The **MaxUserPort** value controls the maximum port number used when an application requests any available user port from the system. Normally, short-lived ports are allocated in the range from 1024 through 5000. Setting this parameter to a value outside of the valid range causes the nearest valid value to be used (5000 or 65534).

MaxUserPort

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Value:	MaxUserPort
Data Type:	REG_DWORD
Range:	5000 to 65534
Default value:	5000
Recommended value:	65534
Value exists by default?	**No**, needs to be added.

The **TcpTimedWaitDelay** value determines the length of time that a connection stays in the TIME_WAIT state when being closed. While a connection is in the TIME_WAIT state, the socket pair cannot be reused. This is also known as the 2MSL state because the value should be twice the maximum segment lifetime on the network. For more information, see Internet RFC 793 (http://go.microsoft.com/fwlink/?LinkId=113719).

TcpTimedWaitDelay

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Value:	TcpTimedWaitDelay
Data Type:	REG_DWORD
Range:	30 to 300
Default value:	240
Recommended value:	30
Value exists by default?	**No**, needs to be added.

The **GlobalMaxTcpWindowSize** value specifies the maximum size of the TCP receive window. The receive window specifies the number of bytes that a sender can transmit without receiving an acknowledgment. In general, larger receive windows improve performance over high-latency, high-bandwidth networks. For greatest efficiency, the receive window should be an even multiple

of the TCP Maximum Segment Size.
The TCP/IP stack of Windows Server 2008 R2 was designed to tune itself in most environments. Instead of using a fixed size for the receive window, TCP negotiates for and adjusts to an even increment of the maximum segment size. Matching the receive window to even increments of the maximum segment size increases the percentage of full-sized TCP segments used during bulk data transmission.

Note

Setting this entry to a value greater than 64 KB can only be achieved when connecting to other systems that support window scaling as described in Internet RFC 1323 (http://go.microsoft.com/fwlink/?LinkId=84406).

GlobalMaxTcpWindowSize

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Value:	GlobalMaxTcpWindowSize
Data Type:	REG_DWORD
Range:	0 to 1073741823
Default value:	This value does not exist by default
Recommended value:	65535
Value exists by default?	**No**, needs to be added.

The **NumTCBTablePartitions** value controls the number of TCB table partitions. The TCB table can be portioned to improve scalability on multi-processor systems by reducing contention on the TCB table. This value should not be modified without a careful performance study. A suggested maximum value is (number of CPUs) * 4 (not counting hyper-threaded CPUs).

NumTCBTablePartitions

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Value:	NumTCBTablePartitions
Data Type:	REG_DWORD
Range:	1 to 65535
Default value:	4
Recommende	Number of physical CPUs or physical CPU cores * 4 (not counting hyper-

d value:	threaded CPUs)
Value exists by default?	**No**, needs to be added.

The **TcpAckFrequency** value specifies the number of ACKs that will be outstanding before the delayed ACK timer is ignored. The **TcpAckFrequency** value can only be set after installing Hotfix 815230, described in Microsoft Knowledge Base article 815230, "Changing the TcpAckFrequency value to 1 does not have any effect" (http://go.microsoft.com/fwlink/?LinkId=124172).

Caution
Do not change the value of this entry before carefully studying the effect of different values in a test environment.

TcpAckFrequency

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Value:	TcpAckFrequency
Data Type:	REG_DWORD
Range:	0 to 255
Default value:	2
Recommended value:	• 5 for 100 MB networks • 13 MB for 1 GB networks
Value exists by default?	**No**, needs to be added.

The **SynAttackProtect** value specifies whether the SYN flooding attack protection feature of TCP/IP is enabled. The SYN flooding attack protection feature of TCP detects symptoms of denial-of-service (DOS) attacks (also known as SYN flooding), and it responds by reducing the time that the server spends on connection requests that it cannot acknowledge. For more information about the **SynAttackProtect** registry entry, see the "Disable Windows Server 2003 Service Pack 1 and Service Pack 2 denial of service checking" section of Optimizing Operating System Performance.

SynAttackProtect

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters

Value:	SynAttackProtect
Data Type:	REG_DWORD
Range:	0 to 1
Default value:	1 for Windows Server 2003 SP1 and later, 0 otherwise
Recommended value:	0 (Only set this on systems with Web exposure if other hardware or software is providing denial of service (DOS) attack protection)
Value exists by default?	**No**, needs to be added.

The **MTU** value specifies the size of the maximum transmission unit (MTU) that TCP/IP uses for the network interface. The value of this entry takes precedence over the MTU that the network adapter detects dynamically. For more information about the MTU value, see Appendix A: TCP/IP Configuration Parameters (http://go.microsoft.com/fwlink/?LinkId=113716).

MTU

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Interfaces*interfaceGUID*
Value:	MTU
Data Type:	REG_DWORD
Range:	88 to the dynamically determined MTU (in bytes)
Default value:	4294967295
Recommended value:	Determine the optimal MTU value as described in the **Find the Optimal MTU** below, under “Applying registry settings with the network optimization Windows PowerShell script”
Value exists by default?	**No**, needs to be added.

The **ForwardBufferMemory** value specifies the size of the buffer that IP allocates for storing packet data in the router packet queue. Because packet queue data buffers are 256 bytes long, the value of this entry must be a multiple of 256.

ForwardBufferMemory

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Value:	ForwardBufferMemory
Data Type:	REG_DWORD
Range:	0 to 4294967295 (bytes, in 256 byte increments
Default value:	74240
Recommended value:	Set to a value 100 * the optimal MTU value as described in the Find the Optimal MTU below, under “Applying registry settings with the network optimization Windows PowerShell script”
Value exists by default?	**No**, needs to be added.

The **MaxForwardBufferMemory** value limits the total amount of memory that IP can allocate to store packet data in the router packet queue

MaxForwardBufferMemory

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Value:	MaxForwardBufferMemory
Data Type:	REG_DWORD
Range:	Dynamically determined MTU to 4294967295 (in bytes)
Default value:	2097152 (bytes)
Recommended value:	Set to a value 100 * the optimal MTU value as described in **Find the Optimal MTU** below, under “Applying registry settings with the network optimization Windows PowerShell script”. This value must be greater than or equal to the value specified for **ForwardBufferMemory**.
Value exists by default?	**No**, needs to be added.

The **NumForwardPackets** value determines the number of IP packet headers that are allocated for the router packet queue. When all headers are in use, the system attempts to allocate more, up to the value configured for MaxNumForwardPackets. This value should be at least as large as the **ForwardBufferMemory** value divided by the maximum IP data size of the networks that are

connected to the router. It should be no larger than the **ForwardBufferMemory** value divided by 256 because at least 256 bytes of forward buffer memory is used for each packet. The optimal number of forward packets for a given **ForwardBufferMemory** size depends on the type of traffic that is carried on the network and is somewhere between these two values. This parameter is ignored and no headers are allocated if routing is not enabled.

NumForwardPackets

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Value:	NumForwardPackets
Data Type:	REG_DWORD
Range:	1 to 4294967295
Default value:	50
Recommended value:	Set to 1/256 of the value specified for **ForwardBufferMemory**
Value exists by default?	**No**, needs to be added.

The **MaxNumForwardPackets** value limits the total number of IP packet headers that can be allocated for the router packet queue. This value must be greater than or equal to the value of the **NumForwardPackets** entry.

MaxNumForwardPackets

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Value:	MaxNumForwardPackets
Data Type:	REG_DWORD
Range:	1 to 4294967295
Default value:	4294967295
Recommended value:	Set to 1/256 of the value specified for **ForwardBufferMemory**
Value exists by default?	**No**, needs to be added.

The **TcpWindowSize** value specifies the maximum size of the TCP receive window. The receive

window specifies the number of bytes that a sender can transmit without receiving an acknowledgment. In general, larger receive windows improve performance over high-latency, high-bandwidth networks. For greatest efficiency, the receive window should be an even multiple of the TCP Maximum Segment Size.
The TCP/IP stack of Windows Server 2003 was designed to tune itself in most environments. Instead of using a fixed size for the receive window, TCP negotiates for and adjusts to an even increment of the maximum segment size (MSS). For more information about MSS, see Internet RFC 879 (http://go.microsoft.com/fwlink/?LinkId=113755). Matching the receive window to even increments of the MSS increases the percentage of full-sized TCP segments used during bulk data transmission.

Note
Setting this entry to a value greater than 64 KB can only be achieved when connecting to other systems that support window scaling as described in Internet RFC 1323 (http://go.microsoft.com/fwlink/?LinkId=84406).

TcpWindowSize

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
Value:	TcpWindowSize
Data Type:	REG_DWORD
Range:	0 to 4294967295 (bytes, in 256-byte increments)
Default value:	The smaller of the following values: • 65535 • The value of the **GlobalMaxTcpWindowSize** registry entry. For more information, see Appendix A: TCP/IP Configuration Parameters (http://go.microsoft.com/fwlink/?LinkId=113716). • 16384 rounded up to an even multiple of the TCP Maximum Segment Size (MSS)
Recommended value:	Value closest to 64420 that is a multiple of the MSS value.
Value exists by default?	**No**, needs to be added.

The **EnableDynamicBacklog** value is a global switch that enables AFD.SYS functionality to withstand large numbers of SYN_RCVD connections efficiently. For more information, see Microsoft Knowledge Base Article 142641, "Internet Server Unavailable Because of Malicious SYN Attacks" (http://go.microsoft.com/fwlink/?LinkId=158218).

EnableDynamicBacklog

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\AFD\Parameters
Value:	EnableDynamicBacklog
Data Type:	REG_DWORD
Range:	0 to 1
Default value:	0
Recommended value:	1
Value exists by default?	**No**, needs to be added.

The **MinimumDynamicBacklog** value spcifies the minimum number of free connections allowed on a listening endpoint. If the number of free connections drops below this value, a thread is queued to create additional free connections.

MinimumDynamicBacklog

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\AFD\Parameters
Value:	MinimumDynamicBacklog
Data Type:	REG_DWORD
Range:	0 to 4294967295
Default value:	This value does not exist by default
Recommended value:	200
Value exists by default?	**No**, needs to be added.

The **MaximumDynamicBacklog** value controls the maximum number of "quasi-free" connections allowed on a listening endpoint. "Quasi-free" connections include the number of free connections plus those connections in a half- connected (SYN_RECEIVED) state. No attempt is made to create additional free connections if doing so would exceed this value.
To take advantage of the changes to Afd.sys, Windows Sockets applications must specifically request a backlog greater than the value configured for **MinimumDynamicBacklog** when they

issue a listen() call. Microsoft applications, such as Internet Information Services (IIS), which has a default backlog of 25, are configurable.

MaximumDynamicBacklog

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\AFD\Parameters
Value:	MaximumDynamicBacklog
Data Type:	REG_DWORD
Range:	0 to 4294967295
Default value:	This value does not exist by default
Recommended value:	20000
Value exists by default?	**No**, needs to be added.

The **DynamicBacklogGrowthDelta** value controls the number of free connections to create when additional connections are necessary. Be careful with this value, as a very large value could lead to explosive free connection allocations.

DynamicBacklogGrowthDelta

Key:	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\AFD\Parameters
Value:	DynamicBacklogGrowthDelta
Data Type:	REG_DWORD
Range:	0 to 4294967295
Default value:	This value does not exist by default
Recommended value:	100
Value exists by default?	**No**, needs to be added.

Applying registry settings with the network optimization Windows PowerShell script

This topic includes a Windows PowerShell script that can be run on each computer in the BizTalk Server environment to apply registry settings that will optimize networking performance, by using the recommended values discussed in this topic. To run this script, follow these steps:

1. **Find the optimal MTU** - Before running the network optimization PowerShell script, the "Maximum Common MTU" for all computers in the solution must be determined. This means that if 1500 works as an MTU value on all but one computer, either the value of that less capable computer must be used everywhere, or the NIC on the less capable computer needs to be replaced with one as capable as the others. To find the optimal MTU size that can be used on a particular network, do the following:
   a. On each computer that is part of the solution, open a PowerShell window or CMD.exe window and ping that computer's default gateway with the following command:

      ```
      PING –f –l <MTU Size> <interface default gateway address>
      ```

      Note
      If you don't know the default gateway IP address, run "IPCONFIG" from a command prompt to display the address.

      If this command returns the message "Packet needs to be fragmented but DF set," then the specified MTU value is too high and a lower value must be used.
   b. Determine the highest MTU value that will work on all computers in the BizTalk Server environment, this value will be passed as an argument to the network optimization PowerShell script.
2. **Run the network optimization script**
   a. Copy the script from the "Optimizing network performance registry settings" section of Windows PowerShell Scripts into notepad and save as Set-NetworkRegSettings.ps1.
   b. Launch PowerShell and change directories to the folder that contains the saved script.
   c. Run the script with the following command:

      ```
      .\Set-NetworkRegSettings.ps1 1400
      ```

      Note
      In this example an "optimal MTU" size of 1400 is passed as an argument to the script but the "optimal MTU" value may be different in your environment.

      Note
      If the script does not run, or opens in Notepad instead of running, ensure the PowerShell execution policy permits running PowerShell scripts. To determine the current PowerShell execution policy run the **Get-ExecutionPolicy** PowerShell command. To change the current PowerShell execution policy run the **Set-ExecutionPolicy** PowerShell command.

   The network optimizations PowerShell script generates a log file named "*COMPUTERNAME*-NetworkRegSettings.log" in the directory from which the script was run. This log file details which values were changed and lists the original value as well as the new value. For simplicity sake, and so that all logs are accessible from the same place, it is recommended that this script is placed in a networked file share and that the script is run from that file share on all computers in the BizTalk Server environment.

See Also

General Guidelines for Improving Network Performance

Optimizing IIS Performance

Apply IIS configuration options to improve IIS performance

Internet Information Services (IIS) exposes numerous configuration parameters that affect IIS performance. This topic describes several of these parameters and provides general guidance for setting the parameter values to improve IIS performance.

Log only essential information or completely disable IIS logging

IIS logging should be minimized or even disabled in a production environment. To disable logging follow these steps:

1. Click **Start**, point to **All Programs**, click **Administrative Tools**, and then click **Internet Information Services (IIS) Manager**.
2. In the **Connections** pane, click to expand **Sites**, click to select the Web site for which you would like to disable logging, click to select **Features View**, and then double-click the **Logging** feature.
3. Click **Disable** in the **Actions** pane to disable logging for this Web site.

Disable IIS ASP debugging in production environments

IIS ASP debugging should be disabled in a production environment. To disable IIS ASP debugging follow these steps:

1. Click **Start**, point to **All Programs**, click **Administrative Tools**, and then click **Internet Information Services (IIS) Manager**.
2. In the **Connections** pane, click to expand **Sites**, click to select the web site for which you would like to disable ASP debugging, click to select **Features View**, and then double-click the **ASP** feature.
3. Click to expand **Compilation**, click to expand **Debugging Properties**, and verify that both **Enable Client-side Debugging** and **Enable Server-side Debugging** are set to **False**.
4. If necessary, click **Apply** in the **Actions** pane.

Disable debugging for ASP.NET Applications and Web Services by specifying the <compilation debug="false"/> section in the web.config file for the web application.

Tune the value of the ASP Threads Per Processor Limit property

The ASP **Threads Per Processor Limit** property specifies the maximum number of worker threads per processor that IIS creates. Increase the value for the Threads Per Processor Limit until the processor utilization meets at least 50 percent or above. This setting can dramatically influence the scalability of your Web applications and the performance of your server in general. Because this property defines the maximum number of ASP requests that can execute simultaneously, this setting should remain at the default value unless your ASP applications are making extended calls to external components. In this case, you may increase the value of

Threads Per Processor Limit. Doing so allows the server to create more threads to handle more concurrent requests. The default value of Threads Per Processor Limit is 25. The maximum recommended value for this property is 100.

To increase the value for the Threads Per Processor Limit follow these steps:

1. Click **Start**, point to **All Programs**, click **Administrative Tools**, and then click **Internet Information Services (IIS) Manager**.
2. In the **Connections** pane, select the web server, click to select **Features View**, and then double-click the **ASP** feature.
3. Click to expand **Limits Properties** under **Behavior**, click **Threads Per Processor Limit**, enter the desired value for **Threads Per Processor Limit** and click **Apply** in the **Actions** pane.

For more information about how to modify the properties in the <limits> element of the IIS 7.5/7.0 <asp> element, see ASP Limits <limits> (http://go.microsoft.com/fwlink/?LinkId=157483).

Note

Because this property can only be applied at the server level, modification of this property affects all Web sites that run on the server.

Tune the value of the ASP Queue Length property

The goal of tuning this property is to ensure good response time while minimizing how often the server sends the HTTP 503 (Server Too Busy) error to clients when the ASP request queue is full. If the value of ASP Queue Length property is too low, the server will send the HTTP 503 error with greater frequency. If the value of ASP Queue Length property is too high, users might perceive that the server is not responding when in fact their request is waiting in the queue. By watching the queue during periods of high traffic, you should discern a pattern of web request peaks and valleys. Make note of the peak value, and set the value of the ASP Queue Length property just above the peak value. Use the queue to handle short-term spikes, ensure response time, and throttle the system to avoid overload when sustained, unexpected spikes occur. If you do not have data for adjusting the ASP Queue Length property, a good starting point will be to set a one-to-one ratio of queues to total threads. For example, if the ASP Threads Per Processor Limit property is set to 25 and you have four processors (4 * 25 = 100 threads), set the ASP Queue Length property to 100 and tune from there.

To increase the value for the Queue Length property follow these steps:

1. Click **Start**, point to **All Programs**, click **Administrative Tools**, and then click **Internet Information Services (IIS) Manager**.
2. In the **Connections** pane, select the Web server, click to select **Features View**, and then double-click the **ASP** feature.
3. Click to expand **Limits Properties** under **Behavior**, click **Queue Length**, enter the desired value for **Queue Length** and then click **Apply** in the **Actions** pane.

For more information about how to modify the properties in the <limits> element of the IIS 7.5/7.0 <asp> element, see ASP Limits <limits> (http://go.microsoft.com/fwlink/?LinkId=157483).

Note

Because this property can only be applied at the server level, modification of this property affects all Web sites that run on the server.

Tune the MaxPoolThreads registry entry

This setting specifies the number of pool threads to create per processor. Pool threads watch the network for requests and process incoming requests. The MaxPoolThreads count does not include threads that are consumed by ISAPI applications. Generally, you should not create more than 20 threads per processor. MaxPoolThreads is a REG_DWORD registry entry located at HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\InetInfo\Parameters\ with a default value of 4.

Disable WCF services tracing

Use the Configuration Editor Tool (SvcConfigEditor.exe) to disable WCF services tracing in a production environment. For more information about the Configuration Editor Tool, see Configuration Editor Tool (SvcConfigEditor.exe) (http://go.microsoft.com/fwlink/?LinkID=127070).

Configure ASP.NET 2.0 MaxConcurrentRequests for IIS 7.5/7.0 Integrated mode

When ASP.NET 2.0 is hosted on IIS 7.5/7.0 in Integrated Mode, the use of threads is handled differently than on IIS 7.5/7.0 in Classic Mode. When ASP.NET 2.0 is hosted on IIS 7.5 in Integrated mode, ASP.NET 2.0 restricts the number of concurrently executing requests instead of the number of threads concurrently executing requests. For synchronous scenarios, this will indirectly limit the number of threads because the number of requests will be the same as the number of threads. But for asynchronous scenarios, the number of requests and threads will likely be very different because you could have far more requests than threads. When you run ASP.NET 2.0 on IIS 7.5 in integrated mode, the minFreeThreads and minLocalRequestFreeThreads of the “httpRuntime” element in the machine.config are ignored. For IIS 7.5 Integrated mode, a DWORD named MaxConcurrentRequestsPerCPU within HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ASP.NET\2.0.50727.0 determines the number of concurrent requests per CPU. By default, the registry key does not exist and the number of requests per CPU is limited to 12. .NET Framework 3.5 SP1 includes an update to the v2.0 binaries that supports configuring IIS application pools via the aspnet.config file. This configuration applies to integrated mode only (Classic/ISAPI mode ignores these settings).The new aspnet.config config section with default values is listed below:

```
<system.web>
   <applicationPool maxConcurrentRequestsPerCPU="12" maxConcurrentThreadsPerCPU="0"
requestQueueLimit="5000"/>
</system.web>
```

In IIS 7.5 Integrated Mode, the maxWorkerThreads and the maxIoThreads parameters in the “processModel” section of the machine.config file are not used to govern the number of running requests, per se, but they are still used to govern the size of the CLR thread pool used by ASP.NET. When the “processModel” section of the machine.config has “autoConfig=true” (which is the default setting), this will give the application pool up to 100 worker threads

(MaxWorkerThreads) per logical CPU. So a common commodity server with 2 dual-core CPUs would have 400 MaxWorkerThreads. This should be sufficient for all but the most demanding applications.

For more information about configuring ASP.NET Thread Usage on IIS 7.5, see Thomas Marquardt's Blog on ASP.NET Thread Usage on IIS 7.0 (http://go.microsoft.com/fwlink/?LinkId=157518).

Configure ASP.NET 4 MaxConcurrentRequests for IIS 7.5/7.0 Integrated mode

With .NET Framework 4, the default setting for maxConcurrentRequestsPerCPU is 5000 which is a very large number and therefore will allow plenty of asynchronous requests to execute concurrently. For more information, see <applicationPool> Element (Web Settings) (http://go.microsoft.com/fwlink/?LinkID=205339).

For IIS 7.5/7.0 Integrated mode, a DWORD named MaxConcurrentRequestsPerCPU within HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ASP.NET\4.0.30319.0 determines the number of concurrent requests per CPU. By default, the registry key does not exist and the number of requests per CPU is limited to 5000.

Enable IIS HTTP compression

To more efficiently use available bandwidth, enable IIS HTTP compression. HTTP compression provides faster transmission time between compression-enabled browsers and IIS, regardless of whether your content is served from local storage or a UNC resource.

- **To configure compression at the Web server level:**
  a. Click **Start**, point to **All Programs**, click **Administrative Tools**, and then click **Internet Information Services (IIS) Manager**.
  b. In the **Connections** pane, select the Web server, click to select **Features View**, and then double-click the **Compression** feature.
  c. Set the desired compression options and then click **Apply** in the **Actions** pane.
- **To configure compression at the Web site level:**
  a. Click **Start**, point to **All Programs**, click **Administrative Tools**, and then click **Internet Information Services (IIS) Manager**.
  b. In the **Connections** pane, click to expand **Sites**, click to select the Web site for which you would like to configure compression, click to select **Features View**, and then double-click the **Compression** feature.
  c. Set the desired compression options and then click **Apply** in the **Actions** pane.

See Also

Optimizing Performance

Optimizing Database Performance

BizTalk Server is an extremely database-intensive application that may require the creation of up to 13 databases in SQL Server. Because one of the primary design goals of BizTalk Server is to

ensure that no messages are lost, BizTalk Server persists data to disk with great frequency and furthermore, does so within the context of an MSDTC transaction. Therefore, database performance is paramount to the overall performance of any BizTalk Server solution.

This section describes general methods for maximizing SQL Server performance as well as methods for maximizing database performance that are specific to a BizTalk Server environment. For additional information about optimizing BizTalk database performance, see the BizTalk Database Optimization whitepaper (http://go.microsoft.com/fwlink/?LinkId=101578).

In This Section

- Pre-Configuration Database Optimizations
- Post-Configuration Database Optimizations
- Optimizing Filegroups for the Databases
- BizTalk Server MessageBox Database Filegroups SQL Script
- Monitoring SQL Server Performance

Pre-Configuration Database Optimizations

Because of the critical role that SQL Server plays in any BizTalk Server environment, it is of paramount importance that SQL Server be configured/tuned for optimal performance. If SQL Server is not tuned to perform well, then the databases used by BizTalk Server will become a bottleneck and the overall performance of the BizTalk Server environment will suffer. This topic describes several SQL Server performance optimizations that should be followed before installing BizTalk Server and configuring the BizTalk Server databases.

Set NTFS File Allocation Unit

SQL Server stores its data in **Extents**, which are collections of eight physically contiguous 8K pages, or 64 KB. Therefore, to optimize disk performance, set the NTFS Allocation Unit size to 64KB as described in the “Disk Configuration Best Practices” section of the SQL Server best practices article “Predeployment I/O Best Practices” (http://go.microsoft.com/fwlink/?LinkId=140818).

Considerations for the version and edition of SQL Server

Various versions and editions of SQL Server provide different features that can affect the performance of your BizTalk Server environment. For example, under high-load conditions, the number of database locks that are available for the 32-bit version of SQL Server might be exceeded, which is detrimental to the performance of the BizTalk solution. Consider housing your MessageBox database on a 64-bit version of SQL Server if you are experiencing "out of lock" errors in your test environment. The number of available locks is significantly higher on the 64-bit version of SQL Server.

Consider the following table when deciding on the database engine features that you will need for your BizTalk environment. For large scale, enterprise-level solutions that require clustering support, BizTalk Server log shipping support, or Analysis Services support, then you will need

SQL Server 2008 R2 or SQL Server 2008 SP1 Enterprise Edition to house the SQL Server databases.

Version and Edition of SQL Server	64-bit support	Multi-Instance Support	Clustering support	Analysis Services
SQL Server 2008 R2 Enterprise Edition	Yes	Yes (50)	Yes	Yes
SQL Server 2008 R2 Standard Edition	Yes	Yes (16)	Yes (2 node)	Yes
SQL Server 2008 R2 Workgroup Edition	Yes	Yes (16)	No	No
SQL Server 2008 SP1 Enterprise Edition	Yes	Yes	Yes	Yes
SQL Server 2008 SP1 Standard Edition	Yes	Yes	Yes (2 node)	Yes
SQL Server 2008 SP1 Workgroup Edition	No	Yes	No	No

For a complete list of the features supported by the editions of SQL Server 2008 R2, see Features Supported by the Editions of SQL Server 2008 R2 (http://go.microsoft.com/fwlink/?LinkId=140465) in the SQL Server 2008 R2 documentation.

Database planning considerations

We recommend that you host your SQL Server databases on fast storage (for example, fast SAN disks or fast SCSI disks). We recommend RAID 10 (1+0) instead of RAID 5 since raid 5 is slower at writing. Newer SAN disks have very large memory caches, so in these cases the raid selection is not as important. To increase performance, databases and their log files can reside on different physical disks.

Also consider tuning the host bus adapter (HBA) queue depth if using a storage area network (SAN). This can significantly impact I/O throughput and out-of-the box values can be insufficient for SQL Server. Testing is required to determine optimal value, although queue depth of 64 is generally accepted as a good starting point in the absence of any specific vendor recommendations

Install the latest service pack and cumulative updates for SQL Server

Install the latest service packs and the latest cumulative updates for SQL Server 2008 R2 and SQL Server 2008 SP1 as well as the latest .NET Framework service packs.

Install SQL Service Packs and cumulative updates on both BizTalk Server and SQL Server

When installing service packs or cumulative updates for SQL Server, also install the service pack or cumulative update on the BizTalk Server computer. BizTalk Server uses SQL Client components that are updated by SQL Server service packs and cumulative updates.

Consider using a fast solid state drive (SSD) to house the SQL Server tembdb

Consider using one or more Solid State Disk (SSD) drives to house the TempDB. SSD drives offer significant performance advantages over traditional hard drives and are quickly dropping in price as they enter mainstream markets. Because TempDB performance is often a key factor for overall SQL Server performance, the added initial cost of the drives will often be quickly recouped by the overall increased SQL Server performance, especially when running enterprise applications for which SQL Server performance is critical.

Consider implementing the SQL Server 2008 R2 Data Collector and Management Data Warehouse

SQL Server 2008 R2 accommodates the use of the new Data Collector and Management Data Warehouse to collect environment/database performance related data for test and trend analysis. The Data Collector persists all collected data to the specified Management Data Warehouse. While this is not a performance optimization this will be useful for analysis of any performance issues.

Grant the account which is used for SQL Server the Windows Lock Pages In Memory privilege

Grant the Windows Lock Pages in Memory privilege to the SQL Server service account. This should be done to prevent the Windows operating system from paging out the buffer pool memory of the SQL Server process by locking memory that is allocated for the buffer pool in physical memory.

In our lab environment, the Windows policy **Lock Pages in Memory** option was enabled by default. For more information about how to enable the **Lock Pages in Memory** option, see How to: Enable the Lock Pages in Memory Option (Windows) (http://go.microsoft.com/fwlink/?LinkID=208267).

Important

Certain limitations apply when granting the SQL Server service account the Windows Lock Pages in Memory privilege. See the following Microsoft Knowledge base articles for more information:

- 918483, “How to reduce paging of buffer pool memory in the 64-bit version of SQL Server 2005” (http://go.microsoft.com/fwlink/?LinkID=148948).

- 970070, “Support for Locked Pages on SQL Server 2005 Standard Edition 64-bit systems and on SQL Server 2008 Standard Edition 64-bit systems” (http://go.microsoft.com/fwlink/?LinkId=160474).

Grant the SE_MANAGE_VOLUME_NAME right to the SQL Server Service Account

Ensure the account running the SQL Server service has the ‘Perform Volume Maintenance Tasks’ Windows privilege or ensure it belongs to a group that does. This will allow instant file Initialization ensuring optimum performance if a database has to Auto-grow.

Set Min and Max Server Memory

The computers running SQL Server that host the BizTalk Server databases should be dedicated to running SQL Server. When the computers running SQL Server that host the BizTalk Server databases are dedicated to running SQL Server, we recommend that the 'min server memory' and 'max server memory' options on each SQL Server instance are set to specify the fixed amount of memory to allocate to SQL Server. In this case, you should set the “min server memory” and “max server memory” to the same value (equal to the maximum amount of physical memory that SQL Server will use). This will reduce overhead that would otherwise be used by SQL Server dynamically managing these values. Run the following T-SQL commands on each computer running SQL Server to specify the fixed amount of memory to allocate to SQL Server:

```
sp_configure ‘Max Server memory (MB)’,(max size in MB)
```

```
sp_configure ‘Min Server memory (MB)’,(min size in MB)
```

Before you set the amount of memory for SQL Server, determine the appropriate memory setting by subtracting the memory required for Windows Server from the total physical memory. This is the maximum amount of memory you can assign to SQL Server.

Note

If the computers running SQL Server that host the BizTalk Server databases also host the Enterprise Single Sign-On Master Secret Server, then you may need to adjust this value to ensure that there is sufficient memory available to run the Enterprise Single Sign-On Service. It is not an uncommon practice to run a clustered instance of the Enterprise Single Sign-On service on a SQL Server cluster to provide high availability for the Master Secret Server. For more information about clustering the Enterprise Single Sign-On Master Secret Server, see the topic How to Cluster the Master Secret Server (http://go.microsoft.com/fwlink/?LinkId=158251) in the BizTalk Server documentation.

Split the tempdb database into multiple data files of equal size on each SQL Server instance used by BizTalk Server

Ensuring that the data files used for the tempdb are of equal size is critical because the proportional fill algorithm used by SQL Server is based on the size of the data files. If data files are created with unequal sizes, the proportional fill algorithm will use the largest file more for GAM allocations rather than spreading the allocations between all the files, thereby defeating the purpose of creating multiple data files. The optimal number of tempdb data files depends on the degree of latch contention seen in tempdb. As a general rule of thumb, the number of data files

should be equal to number of processor cores/CPUs where number of CPUs is 8 or less. For servers with more than 8 CPUs, create data files for half the number of CPUs (again, only you have latch contention).

In our lab environment, we used the script below to create 8 TempDB data files each of which had a file size of 1024 MB with 100 MB growth and a log file of 512 MB with 100 MB growth. The data files are moved to drive H: and log file are moved to drive I:.

Important
This script is provided “as is,” is intended for demo or educational purposes only, and is to be used at your own risk. Use of this script is not supported by Microsoft, and Microsoft makes no guarantees about the suitability of this script.

```
--<<<<<<<<<<----------------------------------------------------------------------->>>>>>>>>>--
-- Use of included script samples are subject to the terms specified at
-- http://www.microsoft.com/info/cpyright.htm
--<<<<<<<<<<----------------------------------------------------------------------->>>>>>>>>>--
--***Instructions***
-- 1. If running the script from a remote server, change the context in SSMS to target
instance
-- 2. Enable SQLCMD mode (add & click toolbar button or toggle by clicking Query > SQLCMD
Mode)
-- 3. Commence execution of scripts (recommend running statements discretely to more
easily remedy potential problems)
-- 4. Examine servername & temp configuration
-- 5. If necessary, 1) Replace instance name in path to reflect target instance *all
throughout script*
      --            2) Modify root drives to reflect drives designated for data & log
(folder creation *and* ALTER DB statements)
-- 6. Resume script execution
-- 7. If necessary, create new folders
-- 8. Modify/Add data & log files
-- 9. Recycle SQL service using sqlservermanager10.msc
--10. Examine results & if appropriate, delete original tempdb data log files
 --(if they were "moved", the original files aren't automatically deleted)


--<<<<<<<<<<----------------------------------------------------------------------->>>>>>>>>>--
--1. If running the script from a remote server, change the context in SSMS to target
instance
```

```
--2. Enable SQLCMD mode (add & click toolbar button or toggle by clicking Query > SQLCMD
Mode)
--3. Commence execution of scripts (recommend running statements discretely to more
easily remedy potential problems)
--4. Examine servername & temp configuration
SELECT @@SERVERNAME
EXEC dbo.sp_helpdb tempdb
--tempdev   1   C:\tempdb.mdf   PRIMARY  8192 KB  Unlimited  10%  data only
--templog   2   C:\templog.ldf  NULL      512 KB  Unlimited  10%  log only
GO
--5. If necessary, 1) Replace instance name in path to reflect target instance *all
throughout script*
     --            2) Modify root drives to reflect drives designated for data & log
(folder creation *and* ALTER DB statements)
--6. Resume script execution
--7. If necessary, create new folders
--!!md H:\MSSQL10.<instance>
--!!md H:\MSSQL10.<instance>\MSSQL
--!!md H:\MSSQL10.<instance>\MSSQL\DATA
GO
-- 8. Modify/Add data & log files
 --note: even if the out-of-box mdf is already where it needs to be,
   --the first command is necessary to modify size & filegrowth
ALTER DATABASE tempdb MODIFY FILE (NAME = tempdev  , FILENAME = 'H:\tempdb.mdf'   , SIZE
= 1024MB , FILEGROWTH = 100MB)
ALTER DATABASE tempdb ADD FILE    (NAME = tempdat2 , FILENAME = 'H:\tempdat2.ndf' , SIZE
= 1024MB , FILEGROWTH = 100MB)
ALTER DATABASE tempdb ADD FILE    (NAME = tempdat3 , FILENAME = 'H:\tempdat3.ndf' , SIZE
= 1024MB , FILEGROWTH = 100MB)
ALTER DATABASE tempdb ADD FILE    (NAME = tempdat4 , FILENAME = 'H:\tempdat4.ndf' , SIZE
= 1024MB , FILEGROWTH = 100MB)
ALTER DATABASE tempdb ADD FILE    (NAME = tempdat5 , FILENAME = 'H:\tempdat5.ndf' , SIZE
= 1024MB , FILEGROWTH = 100MB)
ALTER DATABASE tempdb ADD FILE    (NAME = tempdat6 , FILENAME = 'H:\tempdat6.ndf' , SIZE
= 1024MB , FILEGROWTH = 100MB)
```

```
ALTER DATABASE tempdb ADD FILE    (NAME = tempdat7 , FILENAME = 'H:\tempdat7.ndf' , SIZE
= 1024MB , FILEGROWTH = 100MB)
ALTER DATABASE tempdb ADD FILE    (NAME = tempdat8 , FILENAME = 'H:\tempdat8.ndf' , SIZE
= 1024MB , FILEGROWTH = 100MB)
GO
ALTER DATABASE tempdb MODIFY FILE (NAME = templog , FILENAME = 'I:\templog.ldf', SIZE =
512MB , FILEGROWTH = 100MB)
GO
--8b. Modify log file:  modify drive & instance name to reflect designated destination
for tempdb log
--!!md I:\MSSQL10.<instance>
--!!md I:\MSSQL10.<instance>\MSSQL
--!!md I:\MSSQL10.<instance>\MSSQL\DATA
GO

-- 9. Recycle SQL service in SQL Server Services node of sqlservermanager10.msc
    --note, if running script from a UNC share, SSMS will report an error,
      --but SQL Server Configuration Manager will open if its location is in %path%
!!sqlservermanager10.msc

--10. Examine results & if appropriate, delete original tempdb data log files
 --(if they were "moved", the original files aren't automatically deleted)
EXEC dbo.sp_helpdb tempdb
--!!del C:\tempdb.mdf
--!!del C:\templog.ldf
GO
```

Use the SQL Server 2008 Activity Monitor or the SQL Server 2005 Performance Dashboard Reports described in Monitoring SQL Server Performance to identify problems with latch contention.

Manually set SQL Server Process Affinity

The Process Affinity option can provide performance enhancements in high-end, enterprise-level SQL Server environments that are running on non-NUMA computers with 16 or more CPUs. This

is especially true in high-throughput BizTalk environments where you have contention on shared tables in the MessageBox database. Because the SQL Server computers that were used in our lab environment were not NUMA-enabled and had 16 cores, to optimize performance, we used the commands below to set process affinity:

To manually set the SQL Server Process Affinity from 0 to 15

```
ALTER SERVER CONFIGURATION

SET PROCESS AFFINITY CPU = 0 to 15
```

For more information, see ALTER SERVER CONFIGURATION (Transact-SQL) (http://go.microsoft.com/fwlink/?LinkID=208269).

Configure MSDTC

To facilitate transactions between SQL Server and BizTalk Server, you must enable Microsoft Distributed Transaction Coordinator (MS DTC). To configure MSDTC on SQL Server, see the topic General Guidelines for Improving Operating System Performance.

Enable Trace Flag T1118 as a startup parameter for all instances of SQL Server

Implementing Trace Flag –T1118 helps reduce contention across the SQL Server instances by removing almost all single page allocations. For more information, see Microsoft Knowledge Base article 328551, "PRB: Concurrency enhancements for the tempdb database" (http://go.microsoft.com/fwlink/?LinkID=153694).

Do not change default SQL Server settings for max degree of parallelism, SQL Server statistics, or database index rebuilds and defragmentation

If a SQL Server instance will house BizTalk Server databases, then there are certain SQL Server settings that should not be changed. Specifically, the SQL Server max degree of parallelism, the SQL Server statistics on the MessageBox database, and the settings for the database index rebuilds and defragmentation should not be modified. For more information, see the topic SQL Server Settings That Should Not Be Changed (http://go.microsoft.com/fwlink/?LinkId=160068) in the BizTalk Server 2010 Operations Guide.

See Also

Optimizing Database Performance

Post-Configuration Database Optimizations

In addition to following the recommendations in Pre-Configuration Database Optimizations, several steps should be followed to optimize BizTalk Server database performance on SQL Server *after* BizTalk Server has been installed and the BizTalk Server databases have been configured. This topic provides a list of these optimizations.

Consider setting the 'text in row' table option on specific MessageBox database tables

SQL Server provides a table option called **text in row** to declare that the contents of the fields of type **text**, **ntext**, or **image** data whose dimensions are smaller than those of a data page (8Kb) must be stored in a data row. By setting this option on BizTalkMsgBoxDb tables (Parts table, Spool table and DynamicStateInfo Tables) , you can increase message throughput when working with small messages which have a small context and orchestrations that have a small persistence size.

- **Parts Table**: When the message size is smaller than the dimensions of a data page that are of 8kb, applying the **text in row** table option on the Parts table can lead to BizTalk Server performance improvement. The Parts table contains the following fields:
  - **ImgPart**: Contains a message part or message part fragment.
  - **ImgPropBag**: Contains the message part property bag.

  This way, when looping against the MessageBox, the Message Agent running within a BizTalk host can retrieve a batch of messages from the Parts table by reading small amount of pages. Depending on the specific scenario and hardware configuration, this technique can reduce CPU utilization both on the SQL Server and BizTalk Server, and provide a significant improvement in terms of latency and throughput.
- **Spool Table**: When the average size of the message context is less than 8 kb, enabling the **text in row** table option on the Spool table helps you reduce the number of accesses when reading messages from the MessageBox along with their context. To apply this option to the Spool table, you must eliminate unnecessary context properties and distinguished fields to reduce the size of the message context lower than 8 Kb.
- **DynamicStateInfo Tables** These tables, one for each host, contain a field of type image called imgData that contains binary-serialized orchestration state when they encounter a persistence point during their execution. When the internal state of orchestrations within a host HostA is so small that its size once serialized is less than 8 kb, the **text in row** technique can successfully be applied to the DynamicStateInfo_HostA table. Therefore we recommend that you keep the internal state of orchestrations as small as possible. This technique can significantly reduce the time that is spent by the XLANG Engine to serialize, persist, de-serialize and restore the internal state of an orchestration in case of persistence point.

We used the following settings in our lab tests:

- EXEC sp_tableoption N'Spool', 'text in row', '6000'
- EXEC sp_tableoption N'Parts', 'text in row', '6000'

Define auto-growth settings for BizTalk Server databases to a fixed value instead of a percentage value

- SQL Server database auto-growth is a blocking operation, which hinders BizTalk Server database performance. Therefore it is important to allocate sufficient space for the BizTalk Server databases in advance to minimize the occurrence of database auto-growth.
- Database auto-growth should be set to a fixed number of megabytes instead of to a percentage (specify file growth **In Megabytes**). This should be done so that, if auto-growth occurs, it does so in a measured fashion. This reduces the likelihood of excessive database

growth. The growth increment for BizTalk Server databases should generally be no lower than 100 MB.

Pre-size BizTalk Server databases to appropriate size with multiple data files

When SQL Server increases the size of a file, it must first initialize the new space before it can be used. This is a blocking operation that involves filling the new space with empty pages. SQL Server 2005 or later running on Windows Server 2003 or later supports “instant file initialization.” This can greatly reduce the performance impact of a file growth operation. For more information, see Database File Initialization (http://go.microsoft.com/fwlink/?LinkId=132063) in the SQL Server books online. This topic provides steps for enabling instant file initialization.

The following list describes the BizTalk Server database configurations used in our lab tests:

- **BizTalk DTADB (BizTalk Tracking database files):** Data file having a file size of 2048 MB with 100 MB growth and a log file of 1024 MB with 100 MB growth.
- **BizTalkMgmtdb (BizTalk Management database files):** Data file having a file size of 512 MB with 100 MB growth and a log file of 512 MB with 100 MB growth.
- **SSODB:** Data file having a file size of 512 MB with 100 MB growth and a log file of 512 MB with 100 MB growth.
- **BizTalkMsgBoxDb (BizTalk MessageBox databases):** 8 data files, each having a file size of 2 GB with 100 MB growth and a log file of 20 GB with 100 MB growth. Because the BizTalk MessageBox databases are the most active, we recommend you place the data files and transaction log files on dedicated drives to reduce the likelihood of problems with disk I/O contention. In our lab environment, we used one drive for each of the following:
  - MessageBox data files
  - MessageBox transaction log files

The following SQL script can be used to pre-size BizTalkMsgBoxDb which initially has a data file on drive J (J:\BizTalkMsgBoxDb.mdf) and a log file on drive K (K:\BizTalkMsgBoxDb_log.LDF):

Important

This script is provided “as is,” is intended for demo or educational purposes only, and is to be used at your own risk. Use of this script is not supported by Microsoft, and Microsoft makes no guarantees about the suitability of this script.

```
EXEC dbo.sp_helpdb BizTalkMsgBoxDb

ALTER DATABASE BizTalkMsgBoxDb MODIFY FILE (NAME = BizTalkMsgBoxDb , FILENAME =
'J:\BizTalkMsgBoxDb.mdf' , SIZE = 2GB , FILEGROWTH = 100MB)

ALTER DATABASE BizTalkMsgBoxDb ADD FILE    (NAME = BizTalkMsgBoxDb_2 , FILENAME =
'J:\BizTalkMsgBoxDb_2.ndf' , SIZE = 2GB , FILEGROWTH = 100MB)

ALTER DATABASE BizTalkMsgBoxDb ADD FILE    (NAME = BizTalkMsgBoxDb_3 , FILENAME =
'J:\BizTalkMsgBoxDb_3.ndf' , SIZE = 2GB , FILEGROWTH = 100MB)

ALTER DATABASE BizTalkMsgBoxDb ADD FILE    (NAME = BizTalkMsgBoxDb_4 , FILENAME =
'J:\BizTalkMsgBoxDb_4.ndf' , SIZE = 2GB , FILEGROWTH = 100MB)
```

```
ALTER DATABASE BizTalkMsgBoxDb ADD FILE    (NAME = BizTalkMsgBoxDb_5 , FILENAME =
'J:\BizTalkMsgBoxDb_5.ndf' , SIZE = 2GB , FILEGROWTH = 100MB)

ALTER DATABASE BizTalkMsgBoxDb ADD FILE    (NAME = BizTalkMsgBoxDb_6 , FILENAME =
'J:\BizTalkMsgBoxDb_6.ndf' , SIZE = 2GB , FILEGROWTH = 100MB)

ALTER DATABASE BizTalkMsgBoxDb ADD FILE    (NAME = BizTalkMsgBoxDb_7 , FILENAME =
'J:\BizTalkMsgBoxDb_7.ndf' , SIZE = 2GB , FILEGROWTH = 100MB)

ALTER DATABASE BizTalkMsgBoxDb ADD FILE    (NAME = BizTalkMsgBoxDb_8 , FILENAME =
'J:\BizTalkMsgBoxDb_8.ndf' , SIZE = 2GB , FILEGROWTH = 100MB)

GO

ALTER DATABASE BizTalkMsgBoxDb MODIFY FILE (NAME = BizTalkMsgBoxDb_log , FILENAME =
'K:\BizTalkMsgBoxDb_log.LDF', SIZE =  20GB , FILEGROWTH = 100MB)

GO
```

Move the Backup BizTalk Server output directory to a dedicated LUN

Move the Backup BizTalk (Full and Log backup) output directory to a dedicated LUN, and then edit the Backup BizTalk Server job to point to the dedicated LUN. Moving the Backup BizTalk Server output directory to a dedicated LUN will reduce disk I/O contention when the job is running by writing to a different disk than the job is reading from. For more information about configuring the Backup BizTalk Server job see How to Configure the Backup BizTalk Server Job (http://go.microsoft.com/fwlink/?LinkID=153813) in BizTalk Server 2010 help.

Verify that the BizTalk Server SQL Agent Jobs are running

BizTalk Server includes several SQL Server Agent jobs that perform important functions to keep your servers operational and healthy. You should monitor the health of these jobs and ensure they are running without errors. One of the most common causes of performance problems in BizTalk Server is the BizTalk Server SQL Agent Jobs are not running, which in turn can cause the MessageBox and Tracking databases to grow unchecked. Follow these steps to ensure the BizTalk Server SQL Agent Jobs are running without problems:

1. **Verify that the SQL Server Agent service is running**.
2. **Verify that the SQL Server Agent jobs installed by BizTalk Server are enabled and running successfully**.
   The BizTalk Server SQL Server Agent jobs are crucial: if they are not running, system performance will degrade over time.
3. **Verify that the BizTalk Server SQL Server Agent jobs are completing in a timely manner**.
   Set up the most recent version of Microsoft System Center Operations Manager to monitor the jobs.
   You should be aware of schedules that are particular to certain jobs:

- The MessageBox_Message_ManageRefCountLog_BizTalkMsgBoxDb job runs continuously by default. Monitoring software should take this schedule into account and not produce warnings.
- The MessageBox_Message_Cleanup_BizTalkMsgBoxDb job is not enabled or scheduled, but it is started by the MessageBox_Message_ManageRefCountLog_BizTalkMsgBoxDb job every 10 seconds. Therefore, this job should not be enabled, scheduled, or manually started.

4. **Verify that the Startup type of the SQL Server Agent service is configured correctly**. Verify the SQL Server Agent service is configured with a **Startup type** of **Automatic** unless the SQL Server Agent service is configured as a cluster resource on a Windows Server cluster. If the SQL Server Agent service is configured as a cluster resource, then you should configure the **Startup type** as **Manual** because the service will be managed by the Cluster service.

Configure Purging and Archiving of Tracking Data

Follow these steps to ensure that purging and archiving of tracking data is configured correctly:

1. Ensure the SQL Agent job “DTA Purge and Archive” is properly configured, enabled, and successfully completing. For more information, see How to Configure the DTA Purge and Archive Job (http://go.microsoft.com/fwlink/?LinkID=153814) in the BizTalk Server documentation.
2. Ensure the job is able to purge the tracking data as fast as the incoming tracking data is generated. For more information, see Measuring Maximum Sustainable Tracking Throughput (http://go.microsoft.com/fwlink/?LinkID=153815) in the BizTalk Server documentation.
3. Review the soft purge and hard purge parameters to ensure you are keeping data for the optimal length of time. For more information, see Archiving and Purging the BizTalk Tracking Database (http://go.microsoft.com/fwlink/?LinkID=153816) in the BizTalk Server documentation.
4. If you only need to purge the old data and do not need to archive it first, change the SQL Agent job to call the stored procedure “dtasp_PurgeTrackingDatabase.” For more information, see How to Purge Data from the BizTalk Tracking Database (http://go.microsoft.com/fwlink/?LinkID=153817) in the BizTalk Server documentation.

Monitor and reduce DTC log file disk I/O contention

The Distributed Transaction Coordinator (DTC) log file can become a disk I/O bottleneck in transaction-intensive environments. This is especially true when using adapters that support transactions, such as SQL Server, MSMQ, or MQSeries, or in a multi-MessageBox environment. Transactional adapters use DTC transactions, and multi-MessageBox environments make extensive use of DTC transactions. To ensure the DTC log file does not become a disk I/O bottleneck, you should monitor the disk I/O usage for the disk where the DTC log file resides on the SQL Server database servers. If disk I/O usage for the disk where the DTC log file resides becomes excessive, consider moving the DTC log file to a faster disk. In an environment where SQL Server is clustered, this is not as much of a concern because the log file will already be on a shared drive, which will likely be a fast SAN drive with multiple spindles. You should nevertheless

still monitor the disk I/O usage. This is because it can become a bottleneck in non-clustered environments or when the DTC log file is on a shared disk with other disk-intensive files.

Separate the MessageBox and Tracking Databases

Because the BizTalk MessageBox and BizTalk Tracking databases are the most active, we recommend you place the data files and transaction log files for each of these on dedicated drives to reduce the likelihood of problems with disk I/O contention. For example, you would need four drives for the MessageBox and BizTalk Tracking database files, one drive for each of the following:

- MessageBox data files
- MessageBox transaction log files
- BizTalk Tracking (DTA) data files
- BizTalk Tracking (DTA) transaction log files

Separating the BizTalk MessageBox and BizTalk Tracking databases and separating the database files and transaction log files on different physical disks are considered best practices for reducing disk I/O contention. Try to spread the disk I/O across as many physical spindles as possible. You can also reduce disk I/O contention by placing the BizTalk Tracking database on a dedicated SQL Server; however, you should still follow the practices above with regards to separating data files and transaction log files.

Optimize filegroups for the BizTalk Server databases

Follow the steps in Optimizing Filegroups for the Databases and the BizTalk Server Database Optimization white paper (http://go.microsoft.com/fwlink/?LinkId=101578) to create additional filegroups and files for the BizTalk Server databases. This will greatly increase the performance of the BizTalk Server databases from a single disk configuration.

See Also

Optimizing Database Performance

Optimizing Filegroups for the Databases

File input/output (I/O) contention is frequently a limiting factor, or bottleneck, in a production BizTalk Server environment. BizTalk Server is a very database intensive application and in turn, the SQL Server database used by BizTalk Server is very file I/O intensive. This topic describes how to make optimal use of the files and filegroups feature of SQL Server to minimize the occurrence of file I/O contention and improve the overall performance of a BizTalk Server solution.

Overview

Every BizTalk Server solution will eventually encounter file I/O contention as throughput is increased. The I/O subsystem, or storage engine, is a key component of any relational database. A successful database implementation typically requires careful planning at the early stages of a project. This planning should include consideration of the following issues:

- What type of disk hardware to use, such as RAID (redundant array of independent disks) devices. For more information about using a RAID hardware solution, see About Hardware-based solutions (http://go.microsoft.com/fwlink/?LinkID=113944) in the SQL Server Books Online.
- How to apportion data on the disks using files and filegroups. For more information about using files and filegroups in SQL Server 2008 R2, see Using Files and Filegroups (http://go.microsoft.com/fwlink/?LinkID=69369) and Understanding Files and Filegroups (http://go.microsoft.com/fwlink/?LinkID=96447) in the SQL Server Books Online.
- Implementing the optimal index design for improving performance when accessing data. For more information about designing indexes, see Designing Indexes (http://go.microsoft.com/fwlink/?LinkID=96457) in the SQL Server Books Online.
- How to set SQL Server configuration parameters for optimal performance. For more information about setting optimal configuration parameters for SQL Server, see Optimizing Server Performance (http://go.microsoft.com/fwlink/?LinkID=71418) in the SQL Server Books Online.

One of the primary design goals of BizTalk Server is to ensure that a message is **never** lost. In order to mitigate the possibility of message loss, messages are frequently written to the MessageBox database as the message is processed. When messages are processed by an orchestration, the message is written to the MessageBox database at every persistence point in the orchestration. These persistence points cause the MessageBox to write the message and related state to physical disk. At higher throughputs, this persistence can result in considerable disk contention and can potentially become a bottleneck.

Making optimal use of the files and filegroups feature in SQL Server has been shown to effectively address File IO bottlenecks and improve overall performance in BizTalk Server solutions.

Note

This optimization should only be done by an experienced SQL Server database administrator and only after all BizTalk Server databases have been properly backed up. This optimization should be performed on all SQL Server computers in the BizTalk Server environment.

SQL Server files and filegroups can be used to improve database performance because this functionality allows a database to be created across multiple disks, multiple disk controllers, or RAID (redundant array of independent disks) systems. For example, if your computer has four disks, you can create a database that is made up of three data files and one log file, with one file on each disk. As data is accessed, four read/write heads can concurrently access the data in parallel. This speeds up database operations significantly. For more information about implementing hardware solutions for SQL Server disks, see Database Performance (http://go.microsoft.com/fwlink/?LinkID=71419) in the SQL Server Books Online.

Additionally, files and filegroups enable data placement, because tables can be created in specific filegroups. This improves performance, because all file I/O for a given table can be directed at a specific disk. For example, a heavily used table can be placed on a file in a

filegroup, located on one disk, and the other less heavily accessed tables in the database can be located on different files in another filegroup, located on a second disk.

File I/O bottlenecks are discussed in considerable detail in the topic Bottlenecks in the Database Tier. The most common indicator that file I/O (disk I/O) is a bottleneck is the value of the “Physical Disk:Average Disk Queue Length” counter. When the value of the “Physical Disk:Average Disk Queue Length” counter is greater than about 3 for any given disk on any of the computers running SQL Server, then file I/O is likely a bottleneck.

If applying file or filegroup optimization doesn't resolve a file I/O bottleneck problem, it may be necessary to increase the throughput of the disk subsystem by adding additional physical or SAN drives.

This topic describes how to manually apply file and filegroup optimizations, but these optimizations can also be scripted. A sample SQL script is provided in BizTalk Server MessageBox Database Filegroups SQL Script.

Note
It is important to note that this script would need to be modified to accommodate the file, filegroup, and disk configuration used by the SQL Server databases for any given BizTalk Server solution.

Note
This topic also describes how to create multiple files and filegroups for the BizTalk MessageBox database. For an exhaustive list of recommended files and filegroups for all of the BizTalk Server databases, see “Appendix B” of the BizTalk Server Database Optimization white paper (http://go.microsoft.com/fwlink/?LinkID=101578).

Note
Even though the BizTalk Server Database Optimization white paper (http://go.microsoft.com/fwlink/?LinkID=101578) was written with BizTalk Server 2006 R2 in mind, the same principles apply to BizTalk Server 2010.

Databases created with a default BizTalk Server configuration

Depending on which features are enabled when configuring BizTalk Server, up to 13 different databases may be created in SQL Server and all of these databases are created in the default filegroup. The default filegroup for SQL Server is the PRIMARY filegroup unless the default filegroup is changed by using the ALTER DATABASE command. The following table lists the databases that are created in SQL Server if all features are enabled when configuring BizTalk Server.

BizTalk Server Databases

Database	Default Database Name	Description
Configuration database	BizTalkMgmtDb	The central meta-information store for all instances of BizTalk Server in the BizTalk

		Server group.
BizTalk MessageBox database	BizTalkMsgBoxDb	Stores subscriptions predicates. It is a host platform, and keeps queues and state tables for each BizTalk Server host. The MessageBox database also stores the messages and message properties.
BizTalk Tracking database	BizTalkDTADb	Stores business and health monitoring data tracked by the BizTalk Server tracking engine.
BAM Analysis database	BAMAnalysis	SQL Server Analysis Services database that keeps the aggregated historical data for Business Activities.
BAM Star Schema database	BAMStarSchema	Transforms the data collected from Business Activity Monitoring for OLAP Processing. This database is required when using the BAM Analysis database.
BAM Primary Import database	BAMPrimaryImport	Stores the events from Business Activities and then queries for the progress and data after activity instances. This database also performs real-time aggregations.
BAM Archive database	BAMArchive	Stores subscription predicates. The BAM Archive database minimizes the accumulation of Business Activity data in the BAM Primary Import database.
SSO database	SSODB	Securely stores the configuration information for receive locations. Stores information for SSO affiliate applications, as well as the

		encrypted user credentials to all the affiliate applications.
Rule Engine database	BizTalkRuleEngineDb	Repository for: • Policies, which are sets of related rules. • Vocabularies, which are collections of user-friendly, domain-specific names for data references in rules.
BizTalk Base EDI database	BizTalkEDIDb	Stores EDI document tracking and processing data.
Human Workflow Services Administration database	BizTalkHwsDb	Stores administrative information required by the BizTalk Human Workflow Services.
Trading Partner Management database	TPM	Stores trading partner data for Business Activity Services (BAS).
Tracking Analysis Server Administration database	BizTalkAnalysisDb	Stores both business and health monitoring OLAP cubes.

Separation of data files and log files

As noted earlier, a default BizTalk Server configuration places the MessageBox Database into a single file in the default filegroup. By default, the data and transaction logs for the MessageBox database are placed on the same drive and path. This is done to accommodate systems with a single disk. A single file/filegroup/disk configuration is **not optimal** in a production environment. For optimal performance, the data files and log files should be placed on separate disks.

Note

Log files are never part of a filegroup. Log space is managed separately from data space.

The 80/20 rule of distributing BizTalk Server databases

The main source of contention in most BizTalk Server solutions, either because of disk I/O contention or database contention, is the BizTalk Server MessageBox database. This is true in both single and multi-MessageBox scenarios. It is reasonable to assume that as much as 80% of the value of distributing BizTalk databases will be derived from optimizing the MessageBox data files and log file. The sample scenario detailed below is focused on optimizing the data files for a

MessageBox database. These steps can then be followed for other databases as needed. For example, if the solution requires extensive tracking, the Tracking database can also be optimized.

Manually adding files to the MessageBox database, step-by-step

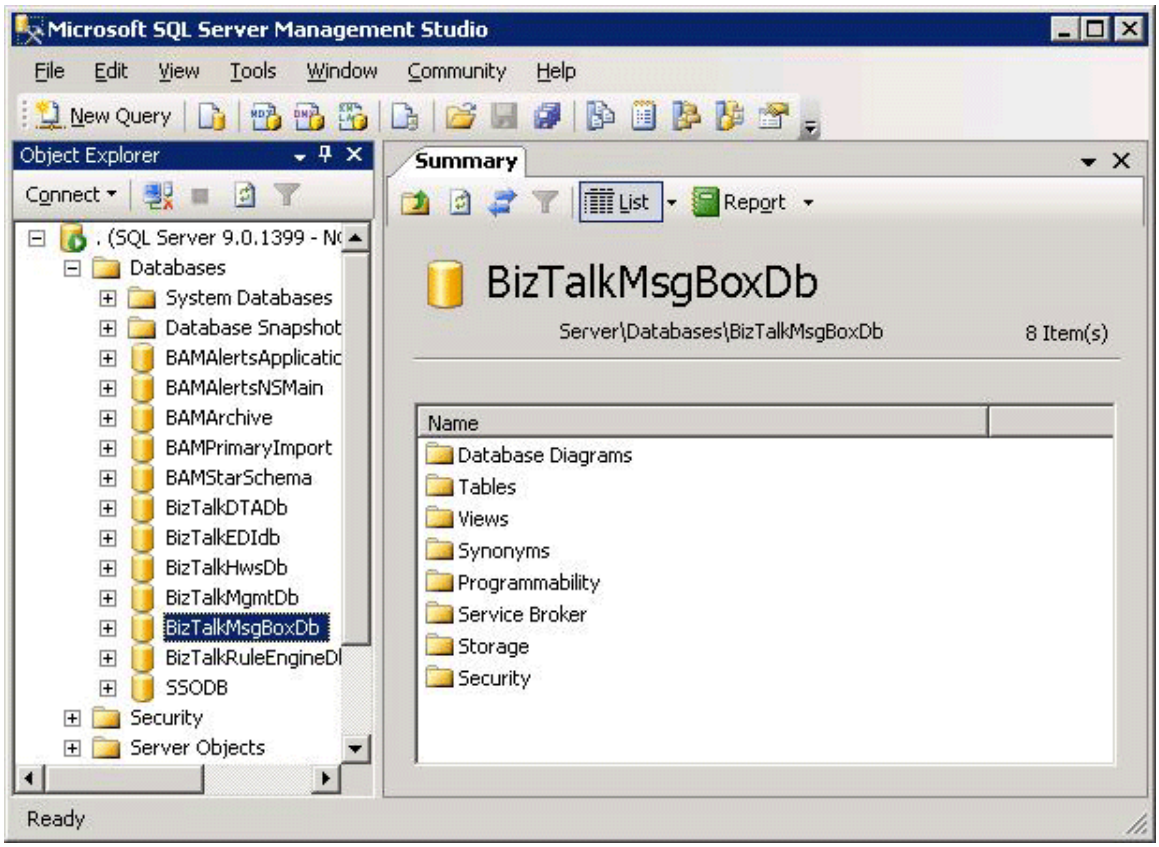
This section of the topic describes the steps that can be followed to manually add files to the MessageBox database. In this example three filegroups are added and then a file is added to each filegroup to distribute the files for the MessageBox across multiple disks. In this example, the steps are performed on SQL Server 2008 R2.

Manually adding files to the MessageBox database on SQL Server 2008 R2

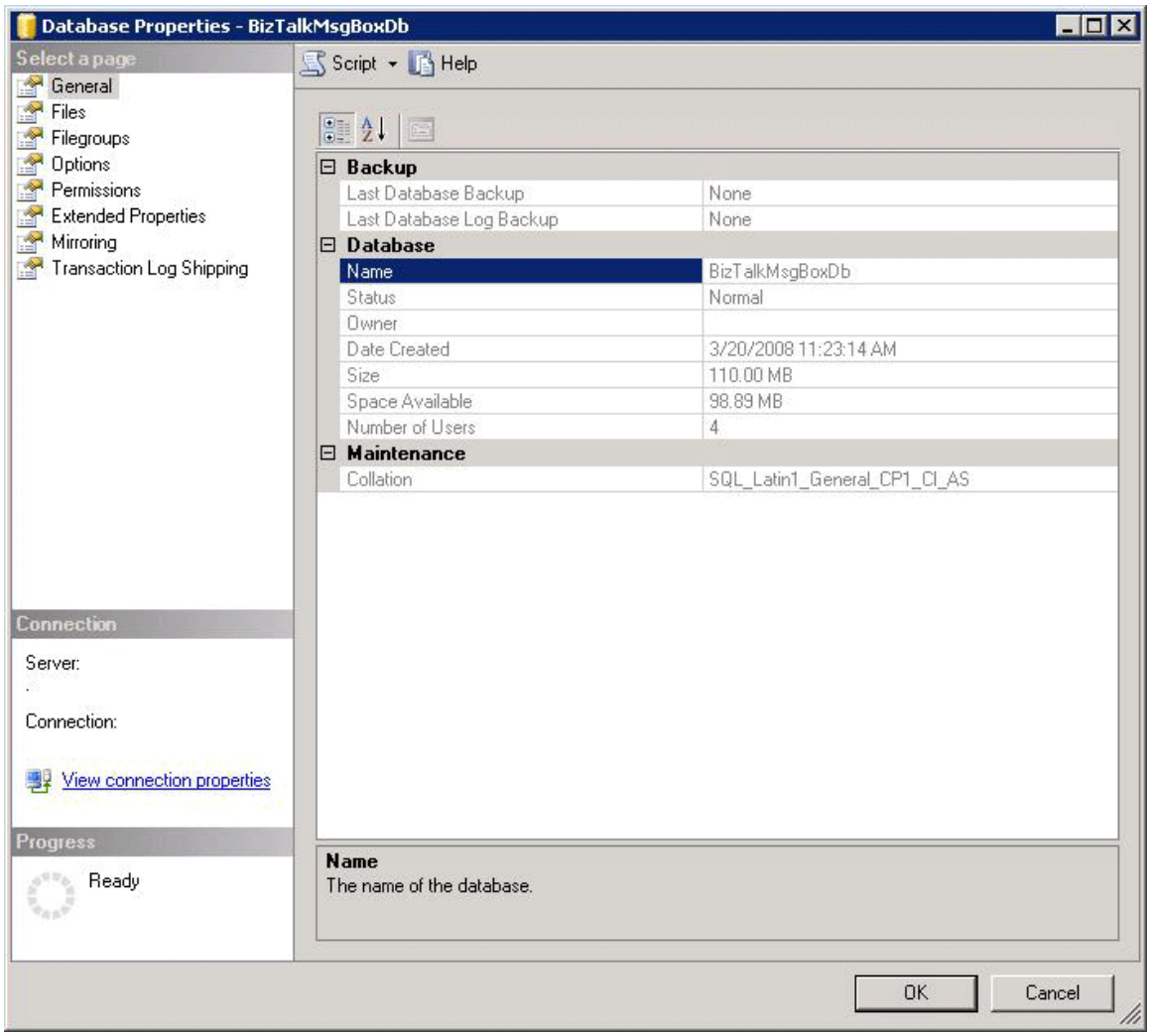
Follow these steps to manually add files to the MessageBox database on SQL Server 2008 R2:

1. Click **Start**, point to **Programs**, point to **SQL Server 2008 R2**, and then click **SQL Server Management Studio** to display the **Connect to Server** dialog box.

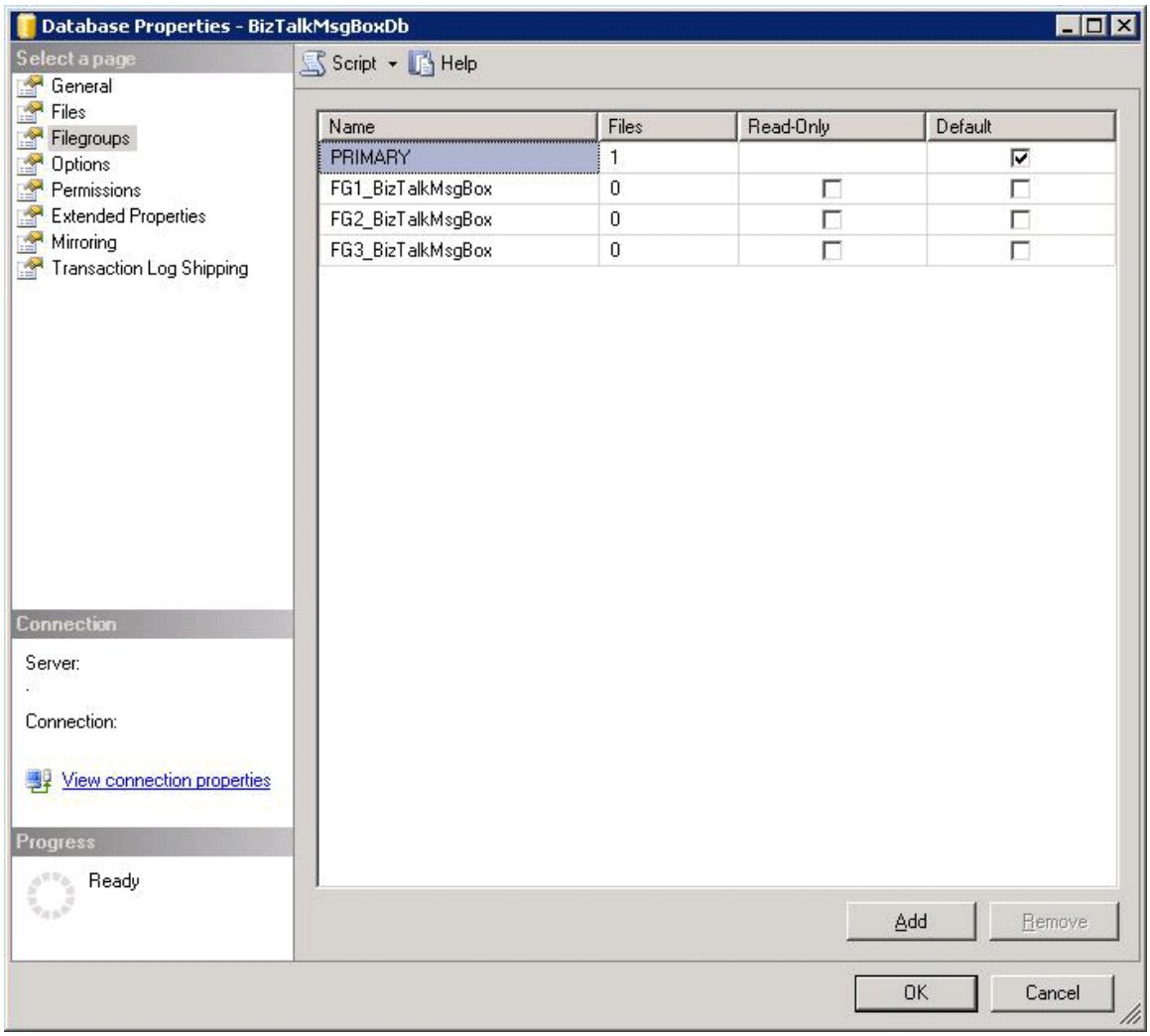
2. In the **Server name** edit box of the **Connect to Server** dialog box, enter the name of the SQL Server instance that houses the BizTalk Server MessageBox databases and click **Connect** to display SQL Server Management Studio. In the **Object Explorer** pane of SQL Server Management Studio, expand **Databases** to view the databases for this instance of SQL Server.



3. Right-click the database to which to add the files, and then click **Properties** to display the **Database Properties** dialog box for the database.



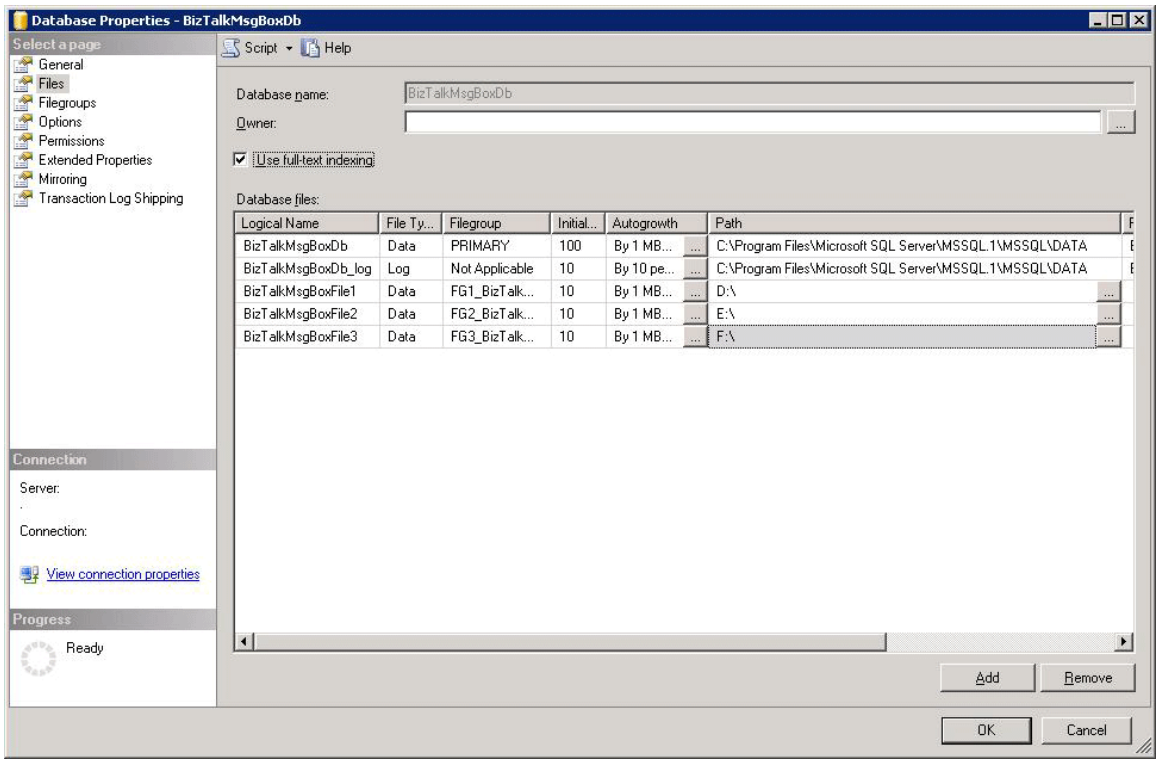
4. In the **Database Properties** dialog box, select the **Filegroups** page. To create additional filegroups for the BizTalkMsgBoxDb databases, click **Add** . In the following example, three additional filegroups are added.



5. In the **Database Properties** dialog box, select the **Files** page.

   To create additional files to add to the filegroups, click **Add** , and then click **OK**. The MessageBox database is now distributed across multiple disks, which will provide a significant performance advantage over a single disk configuration.

   In the following example, a file is created for each of the filegroups that were created earlier and each file is placed on a separate disk.



Sample SQL script for adding filegroups and files to the BizTalk MessageBox database

This guide includes an SQL script for adding filegroups and files to the BizTalk Server MessageBox database.

Note

Because SQL Server writes to its log files sequentially, there is no performance advantage realized by creating multiple log files for a SQL Server database.

To run this script, follow these steps:

1. Click **Start**, point to **Programs**, point to **SQL Server 2008 R2**, and then click **SQL Server Management Studio** to display the **Connect to Server** dialog box.
2. In the **Server name** edit box of the **Connect to Server** dialog box, enter the name of the SQL Server instance that houses the BizTalk Server MessageBox databases and click **Connect** to display the SQL Server Management Studio dialog box.
3. In SQL Server Management Studio, click the **File** menu, point to **New**, and then click **Query with Current Connection** to start the SQL Query Editor.
4. Copy the sample script from BizTalk Server MessageBox Database Filegroups SQL Script into the Query Editor.
5. Edit the parameters in the script to match your BizTalk Server environment, and execute the script.

The advantage to scripting is that scripts can perform multiple tasks quickly, can be reproduced precisely, and reduce the possibility of human error. The disadvantage of scripting is that the

execution of an incorrectly written script can potentially cause serious problems that could require the BizTalk Server databases to be re-configured from scratch.

Important
It is of utmost importance that SQL scripts such as the sample script in this guide are thoroughly tested before being executed in a production environment.

See Also

Optimizing Database Performance

BizTalk Server MessageBox Database Filegroups SQL Script

This topic provides a SQL script that can be run on the SQL Server instances in a BizTalk Server environment to create multiple files and filegroups for the BizTalk MessageBox databases.

Important
This script is provided “as is,” is intended for demo or educational purposes only, and is to be used at your own risk. Use of this script is not supported by Microsoft, and Microsoft makes no guarantees about the suitability of this script.

Important
The following considerations apply when using this SQL script to create multiple files and filegroups for the BizTalk MessageBox databases:

1. **Rerun the MessageBox database filegroups SQL script under the following conditions:**
   - If you install a BizTalk Server hotfix or service pack that runs **msgboxlogic.sql**, you will need to run the MessageBox database filegroups SQL script again. This is necessary because msgboxlogic.sql reverts the MessageBox filegroups and files to default settings, which is to use the PRIMARY filegroup. To determine if a hotfix or service pack runs msgboxlogic.sql, check the **File Information** section of the hotfix KB article. Or check the setup.xml file that is included with the service pack files.
   - If you add a new host to the BizTalk Server group, you will need to run the MessageBox database filegroups SQL script again. This is necessary because the stored procedure that creates new hosts configures the tables for the hosts to use the PRIMARY file group by default.
2. **Applying the MessageBox database filegroups SQL script in a multi-MessageBox environment:** Though not a requirement, the MessageBox database filegroups SQL script can be executed against each MessageBox in a multi-Messagebox environment.

BizTalk MessageBox database filegroups SQL script

The following SQL script can be used to create multiple files and filegroups as described in the topic Optimizing Filegroups for the Databases.

```
/*****************************************************

This script will create multiple filegroups / files for the

BizTalk MessageBox database. It will also set the initial
```

```
size, location, and the number of files per filegroup before
distributing objects among them.

************************************************************/

USE BizTalkMsgBoxDb
GO
SET NOCOUNT ON
GO

/************************************************************
Declare Variables
************************************************************/
DECLARE
      @FILES_PER_FILEGROUP              tinyint,
      @INITIAL_FILE_SIZE                nvarchar(50),
      @GROWTH_INCREMENT                 nvarchar(50),
      @MISC_DATA_FILE_PATH              nvarchar(200),
      @MISC_IDXS_FILE_PATH              nvarchar(200),
      @PREDICATE_DATA_FILE_PATH         nvarchar(200),
      @PREDICATE_IDXS_FILE_PATH         nvarchar(200),
      @MESSAGE_DATA_FILE_PATH           nvarchar(200),
      @MESSAGE_INSTANCE_FILE_PATH       nvarchar(200),
      @MESSAGE_IDXS_FILE_PATH           nvarchar(200),
      @SQLCmd                           nvarchar(2000),
      @DBNAME                           sysname,
      @OBJECT_ID                        int,
      @UNDO                             bit,
      @PRIMARY_FILE_GROUP               sysname,
      @FILE_GROUP                       sysname,
      @FILE_GROUP_ID                    int,
      @FILE_PATH                        nvarchar(200),
      @DEFAULT_FILE_PATH                nvarchar(200),
      @FILE_NAME                        nvarchar(200),
```

```
      @FULL_NAME                        nvarchar(200),
      @FILE_COUNT                       tinyint,
      @COUNTER                          int,
      @DEFAULT_FILEGROUP                nvarchar(50)

/************************************************************
Declare/Create Tables
************************************************************/
Declare @FILE_GROUP_TABLE table
          (
          name sysname,
          file_count tinyint,
          file_path nvarchar(1024)
          )
Declare @TABLE_ALLOCATIONS table
          (
          table_name sysname,
          main_file_group nvarchar(1024),
          ncindex_file_group nvarchar(1024)
          )

/************************************************************
Set User-defined Variables
************************************************************/
--Set to 0, this varibale has no effect,
--Set to 1 to undo changes and move all objects back to Primary filegroups, and delete
all empty data files/filegroups
SET @UNDO = 0
--Set to the number of data files to be created per filegroup
SET @FILES_PER_FILEGROUP = 1
--Set to the initial file size per data file in MB - eg: '100MB'
SET @INITIAL_FILE_SIZE = N'50MB'
--Set to the Growth Increment per data file in MB - eg: '100MB'
SET @GROWTH_INCREMENT = N'50MB'
```

```
--The following 7 Variables should be set to the paths for each filegroup - eg: C:\Data\
SET @MISC_DATA_file_path ='C:\DATA\'
SET @MISC_IDXS_file_path ='C:\DATA\'
SET @PREDICATE_DATA_file_path ='C:\DATA\'
SET @PREDICATE_IDXS_file_path ='C:\DATA\'
SET @MESSAGE_DATA_file_path     ='C:\DATA\'
SET @MESSAGE_INSTANCE_file_path     ='C:\DATA\'
SET @MESSAGE_IDXS_file_path     ='C:\DATA\'
--Set Database Name Variable
SET @DBNAME = DB_NAME()
--Set Default FileGroup Variable
SET @DEFAULT_FILEGROUP = 'MISC_DATA'


/************************************************************
If resetting the object distribution, set the Primary
filegroup as the default
************************************************************/
If @UNDO = 1
BEGIN
     if not exists(SELECT TOP 1 groupname FROM  sys.sysfilegroups WHERE (status & 0x10 >
0) and groupname = 'PRIMARY')
     BEGIN
          SET @DEFAULT_FILEGROUP = 'PRIMARY'
          exec ('ALTER DATABASE ' + @DBNAME + ' MODIFY FILEGROUP [' + @DEFAULT_FILEGROUP
+ '] DEFAULT')
     END
END

/************************************************************
Find path of data file in the default filegroup to use
until we set our user-defined paths - used when 'Undoing'
************************************************************/
```

```
SELECT TOP 1 @PRIMARY_FILE_GROUP = sfg.groupname, @FILE_NAME = sf.name, @FULL_NAME =
sf.filename
FROM sys.sysfiles sf, sys.sysfilegroups sfg
WHERE (sfg.status & 0x10 > 0)
     AND sf.groupid = sfg.groupid

--Find and set our default file path variable
SET @COUNTER = CHARINDEX(@FILE_NAME, @FULL_NAME)
SET @DEFAULT_FILE_PATH = SUBSTRING(@FULL_NAME, 1, @COUNTER - 1)

/*********************************************************
Populate a temporary table with a list of filegroups
we want to create, their paths and the number
of data files to create per filegroup
*********************************************************/
INSERT INTO @file_group_table (name, file_count, file_path) VALUES (N'MISC_DATA',
@FILES_PER_FILEGROUP, @MISC_DATA_file_path)
INSERT INTO @file_group_table (name, file_count, file_path) VALUES (N'MISC_INDEXES',
@FILES_PER_FILEGROUP, @MISC_IDXS_file_path)
INSERT INTO @file_group_table (name, file_count, file_path) VALUES (N'PREDICATE_DATA',
@FILES_PER_FILEGROUP, @PREDICATE_DATA_file_path)
INSERT INTO @file_group_table (name, file_count, file_path) VALUES (N'PREDICATE_INDEXES',
@FILES_PER_FILEGROUP, @PREDICATE_IDXS_file_path)
INSERT INTO @file_group_table (name, file_count, file_path) VALUES (N'MESSAGE_DATA',
@FILES_PER_FILEGROUP, @MESSAGE_DATA_file_path)
INSERT INTO @file_group_table (name, file_count, file_path) VALUES (N'MESSAGE_INSTANCES',
@FILES_PER_FILEGROUP, @MESSAGE_INSTANCE_file_path)
INSERT INTO @file_group_table (name, file_count, file_path) VALUES (N'MESSAGE_INDEXES',
@FILES_PER_FILEGROUP, @MESSAGE_IDXS_file_path)

/*********************************************************
Populate a temporary table with a list of
table and filegroup associations
*********************************************************/
```

```
INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'ActiveRefCountLog', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'AddRef', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'ApplicationProps', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'Applications', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'BitwiseANDPredicates', N'PREDICATE_DATA', N'PREDICATE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'BizTalkDBVersion', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'ConvoySetInstances', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'ConvoySets', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'EqualsPredicates', N'PREDICATE_DATA', N'PREDICATE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'EqualsPredicates2ndPass', N'PREDICATE_DATA', N'PREDICATE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'ExistsPredicates', N'PREDICATE_DATA', N'PREDICATE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'FirstPassPredicates', N'PREDICATE_DATA', N'PREDICATE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'Fragments', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'GreaterThanOrEqualsPredicates', N'PREDICATE_DATA', N'PREDICATE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'GreaterThanPredicates', N'PREDICATE_DATA', N'PREDICATE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'Instances', N'MESSAGE_INSTANCES', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'InstancesOperatedOn', N'MESSAGE_INSTANCES', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'InstancesPendingOperations', N'MESSAGE_INSTANCES', N'MESSAGE_INDEXES')
```

```
INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'InstancesSuspended', N'MESSAGE_INSTANCES', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'LessThanOrEqualsPredicates', N'PREDICATE_DATA', N'PREDICATE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'LessThanPredicates', N'PREDICATE_DATA', N'PREDICATE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'LocalizedErrorStrings', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'MarkLog', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'MessageParts', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'MessagePredicates', N'PREDICATE_DATA', N'PREDICATE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'MessageProps', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'MessageRefCountLog1', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'MessageRefCountLog2', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'MessageRefCountLogTotals', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'MessageZeroSum', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'Modules', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'NotEqualsPredicates', N'PREDICATE_DATA', N'PREDICATE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'OperationsProgress', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'PartRefCountLog1', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'PartRefCountLog2', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'PartRefCountLogTotals', N'MISC_DATA', N'MISC_INDEXES')
```

```
INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'Parts', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'PartZeroSum', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'PredicateGroup', N'PREDICATE_DATA', N'PREDICATE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'PredicateGroupNames', N'PREDICATE_DATA', N'PREDICATE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'PredicateGroupZeroSum1', N'PREDICATE_DATA', N'PREDICATE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'PredicateGroupZeroSum2', N'PREDICATE_DATA', N'PREDICATE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'ProcessHeartbeats', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'Release', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'ServiceClasses', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'Services', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'Spool', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'StaticStateInfo', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'Subscription', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'Tracking_Fragments1', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'Tracking_Fragments2', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'Tracking_Parts1', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'Tracking_Parts2', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'Tracking_Spool1', N'MESSAGE_DATA', N'MESSAGE_INDEXES')
```

```
INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'Tracking_Spool2', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'TrackingData_0_0', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'TrackingData_0_1', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'TrackingData_0_2', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'TrackingData_0_3', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'TrackingData_1_0', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'TrackingData_1_1', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'TrackingData_1_2', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'TrackingData_1_3', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'TrackingDataPartitions', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'TrackingMessageReferences', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'TrackingSpoolInfo', N'MESSAGE_DATA', N'MESSAGE_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'TruncateRefCountLog', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'TrustedUsers', N'MISC_DATA', N'MISC_INDEXES')

INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group) VALUES
(N'UniqueSubscription', N'MISC_DATA', N'MISC_INDEXES')


/************************************************************

Find our BizTalk Host specific tables and add them into

our temporary table

************************************************************/

declare @nvcAppName nvarchar(256)
```

```
declare applications_cursor cursor FAST_FORWARD FOR
SELECT nvcApplicationName FROM Applications

OPEN applications_cursor
FETCH NEXT FROM applications_cursor INTO @nvcAppName
WHILE (@@FETCH_STATUS = 0)
BEGIN
     INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group)
VALUES ('[' + @nvcAppName + N'Q]', N'MISC_DATA', N'MISC_INDEXES')
     INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group)
VALUES ('[' + @nvcAppName + N'Q_Scheduled]', N'MISC_DATA', N'MISC_INDEXES')
     INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group)
VALUES ('[' + @nvcAppName + N'Q_Suspended]', N'MISC_DATA', N'MISC_INDEXES')
     INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group)
VALUES ('[InstanceStateMessageReferences_' + @nvcAppName + N']', N'MISC_DATA',
N'MISC_INDEXES')
     INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group)
VALUES ('[DynamicStateInfo_' + @nvcAppName + N']', N'MISC_DATA', N'MISC_INDEXES')
     INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group)
VALUES ('[' + @nvcAppName + N'_MessageRefCountLog]', N'MISC_DATA', N'MISC_INDEXES')
     INSERT INTO @table_allocations (table_name, main_file_group, ncindex_file_group)
VALUES ('[' + @nvcAppName + N'_DequeueBatches]', N'MISC_DATA', N'MISC_INDEXES')

     FETCH NEXT FROM applications_cursor INTO @nvcAppName
END
CLOSE applications_cursor
DEALLOCATE applications_cursor

/*********************************************************
Loop through our temporary table of filegroups
and create them
*********************************************************/
DECLARE file_group_cursor CURSOR FAST_FORWARD FOR
SELECT name, file_count, file_path FROM @file_group_table
```

```
OPEN file_group_cursor
FETCH NEXT FROM file_group_cursor INTO @file_group, @file_count, @file_path
WHILE ( (@@FETCH_STATUS = 0) AND (@UNDO = 0) )
BEGIN
     SET @file_group_id = FILEGROUP_ID(@file_group)

     /************************************************************
     Create Filegroup if it does not already exist
     ************************************************************/
     IF (@file_group_id IS NULL)
     BEGIN
          exec('ALTER DATABASE [' + @DBNAME + '] ADD FILEGROUP [' + @file_group + ']')
     END

     print 'Creating FileGroup: ' + @file_path + @file_group

     /************************************************************
     Loop through each new filegroup and create the data file
     for the filegroup if one does not exist
     ************************************************************/
     SET @file_group_id = FILEGROUP_ID(@file_group)
     IF NOT EXISTS (SELECT TOP 1 groupid FROM sys.sysfiles WHERE groupid =
@file_group_id)
     BEGIN
          set @COUNTER = 0
          WHILE (@COUNTER < @file_count)
          BEGIN
               if @COUNTER = 0
               BEGIN
                    SET @file_name = @DBNAME + '_' + @file_group
               END
               ELSE
                    SET @file_name = @DBNAME + '_' + @file_group + '_' + CAST(@COUNTER as
nvarchar(3))
```

```
            SET @full_name = @file_path + @file_name + '.ndf'

            print 'Creating Data File: ' + @full_name + ' in ' + @file_group

            --Build our TSQL string to add our data file to the database
            SET @SQLCmd = N'ALTER DATABASE [' + @DBNAME + '] ADD FILE (
                                    NAME = N''' + @file_name + ''',
                                    FILENAME = N''' + @full_name + ''' ,
                                    SIZE = ' + @INITIAL_FILE_SIZE + ',
                                    MAXSIZE = UNLIMITED,
                                    FILEGROWTH = '+ @GROWTH_INCREMENT +' )
                                TO FILEGROUP [' + @file_group + ']'
            --Execute our TSQL string
            EXECUTE (@SQLCmd)

            /*********************************************************
            Set the requested Filegroup to be the default
            *********************************************************/
            If @file_group = @DEFAULT_FILEGROUP
            Begin
                 --Build our TSQL string to set requested FileGroup as default
                 SET @SQLCmd = N'ALTER DATABASE [' + @DBNAME + '] MODIFY FILEGROUP ' +
@file_group + ' DEFAULT'
                 print 'Changing ' + @file_group + ' FileGroup to DEFAULT'
                 --Execute our TSQL string
                 EXECUTE (@SQLCmd)
            END

            set @COUNTER = @COUNTER + 1
       END
  END

  FETCH NEXT FROM file_group_cursor INTO @file_group, @file_count, @file_path
```

```
END
CLOSE file_group_cursor


/************************************************************
Loop through our temporary table of table and filegroup
allocations and recreate clustered and non-clustered indexes
on the New Filegroups as well as move heaps to their new
filegroup.
************************************************************/
declare @table_name sysname,
          @index_name sysname,
          @column_name sysname,
          @column_order int,
          @index_id int,
          @is_unique bit,
          @data_space_id int,
          @ignore_dup_key bit,
          @is_primary_key bit,
          @is_unique_constraint bit,
          @fill_factor tinyint,
          @is_padded bit,
          @is_disabled bit,
          @allow_row_locks bit,
          @allow_page_locks bit,
          @ncindex_file_group nvarchar(1024),
          @ncindex_file_group_id int,
          @create_index_sql nvarchar(4000)

DECLARE table_allocations_cursur CURSOR FAST_FORWARD FOR
SELECT table_name, main_file_group, ncindex_file_group FROM @table_allocations
OPEN table_allocations_cursur
FETCH NEXT FROM table_allocations_cursur INTO @table_name, @file_group,
@ncindex_file_group
```

```
WHILE (@@FETCH_STATUS = 0)
BEGIN
          /************************************************************
          If reverting back to single filegroup, then overwrite
          filegroups with 'PRIMARY'
          ************************************************************/
          if (@UNDO = 1)
          BEGIN
               set @file_group = @primary_file_group
               set @ncindex_file_group = @primary_file_group
          END
          else if (@ncindex_file_group is null)
          BEGIN
               SET @ncindex_file_group = @file_group
          END

          set @file_group_id = FILEGROUP_ID(@file_group)
          set @ncindex_file_group_id = FILEGROUP_ID(@file_group)
          select @object_id = OBJECT_ID(@table_name)

          /************************************************************
          Get data on the clustered and non clustered indexes
          for the current table.
          ************************************************************/
          IF (@object_id is not null)
          BEGIN
               DECLARE table_indexes_cursor CURSOR FAST_FORWARD FOR
               SELECT name, index_id, is_unique, data_space_id, ignore_dup_key,
is_primary_key, is_unique_constraint, fill_factor, is_padded, is_disabled,
allow_row_locks, allow_page_locks
                    FROM sys.indexes WHERE object_id = @object_id AND (type = 0 OR type =
1 OR type = 2)
                    ORDER BY name, index_id ASC
```

```
            OPEN table_indexes_cursor
            FETCH NEXT FROM table_indexes_cursor INTO @index_name, @index_id,
@is_unique, @data_space_id, @ignore_dup_key, @is_primary_key, @is_unique_constraint,
@fill_factor, @is_padded, @is_disabled, @allow_row_locks, @allow_page_locks
            WHILE (@@FETCH_STATUS = 0)
            BEGIN
                  /************************************************************
                  Are we dealing with a HEAP? ie Index_id = 0
                  If so, we need to create a temporary clustered index on
                  new filegroup to move the table before deleting this
                  clustered index
                  ************************************************************/
                  if @index_id = 0
                  BEGIN
                        /************************************************************
                        If dealing with a HEAP...
                        Create temp Clustered Index to move data
                        ************************************************************/
                        SET @create_index_sql = 'CREATE CLUSTERED INDEX TEMP_CIX ON ' +
@table_name + '('
                        select @column_name = name from sys.columns where object_id =
@object_id and column_id = 1
                        SET @create_index_sql = @create_index_sql + @column_name + ') ON
[' + @file_group + ']'
                        print 'Create TEMP Clustered index TEMP_CIX on table ' +
@table_name + ' on FileGroup ' + @file_group
                        exec (@create_index_sql)

                        /************************************************************
                        Delete temp Clustered Index
                        ************************************************************/
                        SET @create_index_sql = 'DROP INDEX TEMP_CIX ON ' + @table_name
                        print 'Drop TEMP Clustered index TEMP_CIX from table
'+@table_name + 'on FileGroup ' + @file_group
                        exec (@create_index_sql)
```

```
                    END

                    /************************************************************
                    Deal with the clustered or non clustered indexes
                    ************************************************************/
                    if ( ( (@index_id = 1) AND (@data_space_id != @file_group_id) ) OR
                          ( (@index_id > 1) AND (@data_space_id !=
@ncindex_file_group_id) ) )
                    BEGIN

                         /************************************************************
                         Build string to recreate this index
                         ************************************************************/
                         SET @create_index_sql = 'CREATE '
                         if (@is_unique = 1)
                              SET @create_index_sql = @create_index_sql + 'UNIQUE '
                         if (@index_id = 1)
                              SET @create_index_sql = @create_index_sql + 'CLUSTERED '
                         else
                              SET @create_index_sql = @create_index_sql + 'NONCLUSTERED '
                         SET @create_index_sql = @create_index_sql + 'INDEX ' +
@index_name + ' ON ' + @table_name + '('

                         /************************************************************
                         Get details of columns for this table
                         ************************************************************/
                         DECLARE index_columns_cursor CURSOR FAST_FORWARD FOR
                         SELECT ic.key_ordinal, c.name FROM sys.columns c
                              JOIN sys.index_columns ic ON ic.object_id = @object_id AND
ic.index_id = @index_id AND c.column_id = ic.column_id
                              WHERE c.object_id = @object_id
                              ORDER BY ic.key_ordinal ASC

                         OPEN index_columns_cursor
```

```
                        FETCH NEXT FROM index_columns_cursor INTO @column_order,
@column_name
                        WHILE (@@FETCH_STATUS = 0)
                        BEGIN
                             if (@column_order > 1)
                                  SET @create_index_sql = @create_index_sql + ', '
                             SET @create_index_sql = @create_index_sql + @column_name
                             FETCH NEXT FROM index_columns_cursor INTO @column_order,
@column_name
                        END
                        CLOSE index_columns_cursor
                        DEALLOCATE index_columns_cursor

                        SET @create_index_sql = @create_index_sql + ') WITH ('
                        if (@ignore_dup_key = 1)
                             SET @create_index_sql = @create_index_sql + 'IGNORE_DUP_KEY
= ON, '
                        if (@allow_page_locks = 0)
                             SET @create_index_sql = @create_index_sql +
'ALLOW_PAGE_LOCKS = OFF, '
                        SET @create_index_sql = @create_index_sql + 'DROP_EXISTING = ON)
'
                        if (@index_id = 1)
                             SET @create_index_sql = @create_index_sql + 'ON [' +
@file_group + ']'
                        else
                             SET @create_index_sql = @create_index_sql + 'ON [' +
@ncindex_file_group + ']'
                        print @create_index_sql
                        --Execute SQL string to recreate the index
                        exec (@create_index_sql)
                   END
                   FETCH NEXT FROM table_indexes_cursor INTO @index_name, @index_id,
@is_unique, @data_space_id, @ignore_dup_key, @is_primary_key, @is_unique_constraint,
@fill_factor, @is_padded, @is_disabled, @allow_row_locks, @allow_page_locks
```

```
               END
               CLOSE table_indexes_cursor
               DEALLOCATE table_indexes_cursor
          END
          FETCH NEXT FROM table_allocations_cursur INTO @table_name, @file_group,
@ncindex_file_group
END


/*************************************************************
If we are setting everything back to a single filegroup
we have to remove previously created files now that they
are empty before we delete the empty filegroups
*************************************************************/
if (@UNDO = 1)
BEGIN
     OPEN file_group_cursor
     FETCH NEXT FROM file_group_cursor INTO @file_group, @file_count, @file_path

     WHILE (@@FETCH_STATUS = 0)
     BEGIN
          SET @file_group_id = FILEGROUP_ID(@file_group)

          IF (@file_group_id IS NOT NULL)
          BEGIN
               DECLARE files_in_group_cursor CURSOR FAST_FORWARD FOR
               SELECT name FROM sys.sysfiles WHERE groupid = @file_group_id
               OPEN files_in_group_cursor
               FETCH NEXT FROM files_in_group_cursor INTO @file_name
               WHILE (@@FETCH_STATUS = 0)
               BEGIN
                    exec('ALTER DATABASE [' + @DBNAME + '] REMOVE FILE [' + @file_name +
']')
                    FETCH NEXT FROM files_in_group_cursor INTO @file_name
```

```
                END

                CLOSE files_in_group_cursor
                DEALLOCATE files_in_group_cursor

                exec('ALTER DATABASE [' + @DBNAME + '] REMOVE FILEGROUP [' + @file_group +
']')
          END

          FETCH NEXT FROM file_group_cursor INTO @file_group, @file_count, @file_path
     END

     /************************************************************
     Truncate T-Log.  WITH TRUNCATE_ONLY not available from SQL Server
     2008 therefore have to alter recovery model to simple,
     issue a checkpoint, then change recovery model back to full
     ************************************************************/
     declare @Currentdbname sysname
     set @Currentdbname = db_name()
     declare @cmd varchar(1000)
     SET @cmd = 'ALTER DATABASE ' + @Currentdbname + ' SET RECOVERY SIMPLE'
     EXEC(@cmd)
     print 'Set Recovery Model to Simple'
     Checkpoint
     print 'Perform CheckPoint'
     SET @cmd = 'ALTER DATABASE ' + @DBNAME + ' SET RECOVERY FULL'
     print 'Set Recovery Model back to Full'
     EXEC(@cmd)

     CLOSE file_group_cursor
END

DEALLOCATE file_group_cursor
GO
```

```
print '********Object to FileGroup Distribution Completed********'


/************************************************************
Verify that Distribution of tables among filegroups has
completed successfully
************************************************************/
SELECT sysobjects.[name] AS TableName, sysindexes.[name], sysfilegroups.groupname AS
FileGroupName,
sysfiles.[filename] AS PhysicalFileName, sysfiles.[name] AS LogicalFileName
FROM sysobjects
JOIN sysindexes ON sysobjects.[id] = sysindexes.[id]
JOIN sysfilegroups ON sysindexes.groupid = sysfilegroups.groupid
JOIN sysfiles ON sysfilegroups.groupid = sysfiles.groupid
where sysobjects.[name] not like 'sys%' and sysobjects.xtype = 'U'
ORDER BY sysobjects.[name], sysindexes.[name]
GO
print '********Object to Filegroup Distribution Report Completed********'
GO
```

See Also

Optimizing Database Performance

Monitoring SQL Server Performance

SQL Server provides several tools for monitoring events in SQL Server and for tuning the physical database design. These tools are described in the topic Tools for Performance Monitoring and Tuning (http://go.microsoft.com/fwlink/?LinkId=146357) in the SQL Server books online. Information about specific tools used for SQL Server performance monitoring and tuning is provided below.

SQL Server Performance Monitoring Tools

Because BizTalk Server is such a database-intensive process, it is often helpful to take a peek at what is going on in SQL Server when troubleshooting BizTalk Server performance issues. The remainder of this topic describes performance monitoring tools available for reviewing both real-time and archived data on SQL Server 2008 R2 and SQL Server 2008 SP1.

SQL Server 2008 R2 Activity Monitor

SQL Server 2008 R2 Activity Monitor provides information about SQL Server processes and how these processes affect the current instance of SQL Server. For more information about SQL

Server 2008 R2 Activity Monitor, see Activity Monitor (http://go.microsoft.com/fwlink/?LinkId=146355) in the SQL Server books online. For information about how to open Activity Monitor from SQL Server Management Studio, see How to: Open Activity Monitor (SQL Server Management Studio (http://go.microsoft.com/fwlink/?LinkId=135094) in the SQL Server books online.

SQL Server 2008 R2 Data Collection

SQL Server 2008 R2 provides a data collector that you can use to obtain and save data that is gathered from several sources. The data collector enables you to use data collection containers, which enable you to determine the scope and frequency of data collection on a computer that is running SQL Server 2008 R2. For more information about implementing SQL Server 2008 R2 data collection, see Data Collection (http://go.microsoft.com/fwlink/?LinkId=146356) in the SQL Server books online.

See Also

Optimizing Database Performance

Optimizing BizTalk Server Performance

This section provides guidelines for improving BizTalk Server performance. The optimizations listed are applied after BizTalk Server has been installed and configured.

In This Section

- General BizTalk Server Optimizations
- Low-Latency Scenario Optimizations
- Low-Latency Scenario Optimizations
- Optimizing Business Activity Monitoring (BAM) Performance
- Optimizing Business Rule Engine (BRE) Performance
- Optimizing BizTalk Server WCF Adapter Performance

General BizTalk Server Optimizations

The following recommendations can be used to increase BizTalk Server performance. The optimizations listed in this topic are applied after BizTalk Server has been installed and configured.

Configure MSDTC

To facilitate transactions between SQL Server and BizTalk Server, you must enable Microsoft Distributed Transaction Coordinator (MS DTC). To configure MSDTC on BizTalk Server, see the topic General Guidelines for Improving Operating System Performance.

Recommendations for configuring BizTalk Server hosts

This section provides recommendations for configuring BizTalk Server hosts.

Create multiple BizTalk Server hosts and separate host instances by functionality

Follow these guidelines to create multiple BizTalk Server hosts and allocate workload across those hosts:

- Create separate hosts for sending, receiving, processing, and tracking functionality. Creating multiple BizTalk hosts provides flexibility when configuring the workload in your BizTalk group and is the primary means of distributing processing across the computers running BizTalk Server in a BizTalk group. Using multiple hosts allows you to stop one host without affecting other hosts. For example, you may want to stop sending messages to let them queue up in the MessageBox database while still allowing a receive adapter running in a different host instance to receive inbound messages. Separating host instances by functionality also provides the following benefits:
  - Running multiple BizTalk hosts reduces contention on the MessageBox database host queue tables because each host is assigned its own work queue tables in the MessageBox database.
  - Throttling is implemented in BizTalk Server at the host-level. This allows you to set different throttling characteristics for each host.
  - Security is implemented at the host-level; each host runs under a discrete Windows identity. This allows you to, for example, give Host_A access to FileShare_B, while not allowing any of the other hosts to access the file share.
- Each host instance has its own set of resources such as memory, handles, and threads in the .NET thread pool. When allocating workload across hosts consider placing resources that scale together in the same host.
- Separate adapters and orchestrations that have different priorities for resources in different hosts. This technique accommodates the placement of hosts running high-priority applications on dedicated BizTalk Server computers.

Note

While there are benefits to creating additional host instances, there are also potential drawbacks if too many host instances are created. Each host instance is a Windows service (BTSNTSvc.exe), which generates additional load against the MessageBox database and consumes computer resources (such as CPU, memory, threads).

For more information about modifying BizTalk Server host properties, see How to Modify Host Properties (http://go.microsoft.com/fwlink/?LinkID=154359) in the BizTalk Server 2010 Help.

Configure a dedicated tracking host

BizTalk Server is optimized for throughput, so the main orchestration and messaging engines do not actually move messages directly to the BizTalk Tracking or BAM databases, as this would divert these engines from their primary job of executing business processes. Instead, BizTalk Server leaves the messages in the MessageBox database and marks them as requiring a move to the BizTalk Tracking database. A background process (the tracking host) then moves the messages to the BizTalk Tracking and BAM databases. Because tracking is a resource-intensive operation, a separate host should be created that is dedicated to tracking, thereby minimizing the impact that tracking has on hosts dedicated to message processing. For optimal performance, there should be least one tracking host instance per MessageBox database. The actual number

of tracking host instances should be N + 1, where N = the number of MessageBox databases. The "+ 1" is for redundancy, in case one of the computers hosting tracking fails.

Using a dedicated tracking host also allows you to stop other BizTalk hosts without interfering with BizTalk Server tracking. The movement of tracking data out of the MessageBox database is critical for a healthy BizTalk Server system. If the BizTalk Host responsible for moving tracking data in the BizTalk group is stopped, the Tracking Data Decode service will not run. The impact of this is as follows:

- HAT tracking data will not be moved from the MessageBox database to the BizTalk Tracking database.
- BAM tracking data will not be moved from the MessageBox database to the BAM Primary Import database.
- Because data is not moved, it cannot be deleted from the MessageBox database.
- When the Tracking Data Decode service is stopped, tracking interceptors will still fire and write tracking data to the MessageBox database. If the data is not moved, this will cause the MessageBox database to become bloated, which will impact performance over time. Even if custom properties are not tracked or BAM profiles are not set up, by default some data is tracked (such as pipeline receive / send events and orchestration events). If you do not want to run the Tracking Data Decode service, turn off all tracking so that no interceptors save data to the database. To disable global tracking, see How to Turn Off Global Tracking (http://go.microsoft.com/fwlink/?LinkID=154193) in BizTalk Server 2010 Help. Use the BizTalk Server Administration console to selectively disable tracking events.

The tracking host should be run on at least two computers running BizTalk Server (for redundancy in case one fails). For optimal performance, you should have at least one tracking host instance per MessageBox database. The actual number of tracking host instances should be (N + 1), where N = the number of MessageBox databases. The "+ 1" is for redundancy, in case one of the computers hosting tracking fails.

A tracking host instance moves tracking data for specific MessageBox databases, but there will never be more than one tracking host instance moving data for a specific MessageBox database. For example, if you have three MessageBox databases, and only two tracking host instances, then one of the host instances needs to move data for two of the MessageBox databases. Adding a third tracking host instance distributes the tracking host work to another computer running BizTalk Server. In this scenario, adding a fourth tracking host instance would not distribute any more tracking host work, but would provide an extra tracking host instance for fault tolerance.

For more information about the BAM Event Bus service, see the following topics in BizTalk Server 2010 Help:

- Managing the BAM Event Bus Service (http://go.microsoft.com/fwlink/?LinkID=154194).
- Creating Instances of the BAM Event Bus Service (http://go.microsoft.com/fwlink/?LinkID=154195).

Do not cluster BizTalk hosts unless absolutely necessary

While BizTalk Server 2010 allows you to configure a BizTalk host as a cluster resource, you should only consider doing this if you need to provide high availability to a resource that cannot be hosted across multiple BizTalk computers. As an example, ports using the FTP adapter should

only reside on one host instance, as the FTP protocol does not provide file locking. However, this introduces a single point of failure, which would benefit from clustering. Hosts that contain adapters, such as file, SQL, HTTP or processing only hosts, can be internally load balanced across computers, and do not benefit from clustering.

Increase the number of HTTP concurrent connections allowed by changing the value for the maxconnection parameter

By default, the HTTP adapters (including WCF-based HTTP adapters) open only two concurrent HTTP connections from each computer running BizTalk Server to any specific destination server.

This setting conforms to the IETF RFC for the HTTP 1.1 specification, and although it is suitable for user scenarios, it is not optimized for high throughput. The setting effectively throttles outbound HTTP calls to each server to two concurrent sends from each computer running BizTalk Server.

To increase the number of concurrent connections, you can modify the value for the maxconnection parameter in the BizTalk Server configuration file, BTSNTSvc.exe.config (or BTSNTSvc64.exe.config for 64-bit hosts), on each BizTalk Server. You can increase this for the specific servers being called. As a rule of thumb, the value for the maxconnection parameter should be set to 12 * the number of CPUs or cores on the computer hosting the web application.

Note

Do not increase the value for the maxconnection parameter to such a large value that the Web server being called is overwhelmed with HTTP connections. After changing the value for the maxconnection parameter, perform stress testing by sending requests to each destination Web server to determine a value for maxconnection that will provide good performance and HTTP sends without overwhelming the target Web servers.

The following is an example of the configuration for the maximum connections property.

```
<configuration>
  <system.net>
    <connectionManagement>
      <add address="www.contoso.com" maxconnection="24" />
      <add address="*" maxconnection="48" />
    </connectionManagement>
  </system.net>
</configuration>
```

For more information about tuning IIS and ASP.NET settings for Web services, see the "ASP.NET settings that can impact HTTP Adapter performance" section of Configuration Parameters that Affect Adapter Performance (http://go.microsoft.com/fwlink/?LinkID=154200) in BizTalk Server 2010 Help.

Manage ASP.NET thread usage or concurrently executing requests for Web applications that can host isolated received locations, back-end Web services and WCF services

The number of worker and I/O threads (IIS 7.5 and IIS 7.0 in classic mode) or the number of concurrently executing requests (IIS 7.5 and 7.0 integrated mode) for an ASP.NET Web application that hosts isolated received locations, back-end Web services and WCF services should be modified under the following conditions:

- CPU utilization is not a bottleneck on the hosting Web server.
- The value of:
  - **ASP.NET Apps v4.0.30319 \Request Wait Time** or **ASP.NET Apps v4.0.30319 \Request Execution Time** performance counters is unusually high.
  - **ASP.NET Apps v2.0.50727\Request Wait Time** or **ASP.NET Apps v2.0.50727\Request Execution Time** performance counters is unusually high.
- An error similar to the following is generated in the Application log of the computer that hosts the Web application.

```
Event Type: Warning
Event Source: W3SVC Event Category: None
Event ID: 1013
Date: 11/4/2010
Time: 1:03:47 PM
User: N/A
Computer: <ComputerName>
Description: A process serving application pool 'DefaultAppPool'
exceeded time limits during shut down. The process id was
'<xxxx>'
```

Manage ASP.NET thread usage for Web applications that can host isolated received locations, back-end Web services and WCF services on IIS 7.5 and IIS 7.0 running in Classic mode

When the **autoConfig** value in the machine.config file of an IIS 7.5 and IIS 7.0 server running in Classic mode is set to **true**, ASP.NET 2.0 and ASP.NET 4 manages the number of worker threads and I/O threads that are allocated to any associated IIS worker processes.

```
<processModel autoConfig="true" />
```

To manually modify the number of worker and I/O threads for an ASP.NET 2.0 and ASP.Net 4 Web application, open the associated machine.config file, and then enter new values for the **maxWorkerThreads** and **maxIoThreads** parameters.

```
<!-- <processModel autoConfig="true" /> -->
    <processModel maxWorkerThreads="200" maxIoThreads="200" />
```

Note

These values are for guidance only; ensure you test changes to these parameters.

For more information about tuning parameters in the machine.config file for an ASP.NET 2.0 Web application, see the Microsoft Knowledge Base article 821268 Contention, poor performance, and deadlocks when you make Web service requests from ASP.NET applications (http://go.microsoft.com/fwlink/?LinkID=144169).

Manage the number of concurrently executing requests for ASP.NET 2.0 Web applications that can host isolated received locations, back-end Web services and WCF services on IIS 7.5 and IIS 7.0 running in Integrated mode

When ASP.NET 2.0 is hosted on IIS 7.5 and IIS 7.0 in integrated mode, the use of threads is handled differently than on IIS 7.5 and IIS 7.0 in classic mode. When ASP.NET 2.0 is hosted on IIS 7.5 and IIS 7.0 in integrated mode, ASP.NET 2.0 restricts the number of concurrently executing **requests** rather than the number of **threads** concurrently executing requests. For synchronous scenarios this will indirectly limit the number of threads but for asynchronous scenarios the number of requests and threads will likely be very different. When running ASP.NET 2.0 on IIS 7.5 and IIS 7.0 in integrated mode, the **maxWorkerThreads** and **maxIoThreads** parameters in the machine.config file are not used to govern the number of running threads. Instead, the number of concurrently executing requests can be changed from the default value of 12 per CPU by modifying the value specified for **maxConcurrentRequestsPerCPU**. The **maxConcurrentRequestsPerCPU** value can be specified either in the reqistry or in the config section of an aspnet.config file. Follow these steps to change the default value for **maxConcurrentRequestsPerCPU** to govern the number of concurrently executing requests:

To set the maxConcurrentRequestsPerCPU value in the registry

Warning

Incorrect use of Registry Editor may cause problems requiring you to reinstall your operating system. Use Registry Editor at your own risk. For more information about how to back up, restore, and modify the registry, see Microsoft Knowledge Base article 256986 Windows registry information for advanced users (http://go.microsoft.com/fwlink/?LinkId=62729).

Note

This setting is global and cannot be changed for individual application pools or applications.

1. Click **Start**, click **Run**, type **regedit.exe**, and then click **OK** to start Registry Editor.
2. Navigate to **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ASP.NET\2.0.50727.0**
3. Create the key by following these steps:
   a. On the **Edit** menu, click **New**, and then click **Key**.
   b. Type **maxConcurrentRequestsPerCPU**, and then press **ENTER**.
   c. Under the **maxConcurrentRequestsPerCPU** key, create a DWORD entry with the new value for maxConcurrentRequestsPerCPU.
   d. Close Registry Editor.

To set the maxConcurrentRequestsPerCPU value for an application pool in the config section of an aspnet.config file

Note

Microsoft .NET Framework 3.5 Service Pack 1 must be installed to accommodate setting the values below via configuration file. You can download Microsoft .NET Framework 3.5 Service Pack 1 from Microsoft .NET Framework 3.5 Service Pack 1 (http://go.microsoft.com/fwlink/?LinkID=136345).

Open the aspnet.config file for the application pool, and then enter new values for the **maxConcurrentRequestsPerCPU** and **requestQueueLimit** parameters.

```
<system.web>

    <applicationPool maxConcurrentRequestsPerCPU="12" requestQueueLimit="5000"/>

</system.web>
```

Note

This value overrides the value specified for **maxConcurrentRequestsPerCPU** in the registry. The requestQueueLimit setting is the same as processModel/requestQueueLimit, except that the setting in the aspnet.config file will override the setting in the machine.config file.

For more information about configuring ASP.NET Thread Usage on IIS 7.0, see Thomas Marquardt's Blog on ASP.NET Thread Usage on IIS 7.0 (http://go.microsoft.com/fwlink/?LinkId=157518).

Manage the number of concurrently executing requests for ASP.NET 4 Web applications that can host isolated received locations, back-end Web services and WCF services on IIS 7.5 and 7.0 running in Integrated mode

With .NET Framework 4, the default setting for maxConcurrentRequestsPerCPU is 5000, which is a very large number and therefore will allow plenty of asynchronous requests to execute concurrently. For more information, see <applicationPool> Element (Web Settings) (http://go.microsoft.com/fwlink/?LinkID=205339).

For IIS 7.5 and IIS 7.0 Integrated mode, a DWORD named MaxConcurrentRequestsPerCPU within HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ASP.NET\4.0.30319.0 determines the number of concurrent requests per CPU. By default, the registry key does not exist and the number of requests per CPU is limited to 5000.

To set the maxConcurrentRequestsPerCPU value in the registry

Warning

Incorrect use of Registry Editor may cause problems requiring you to reinstall your operating system. Use Registry Editor at your own risk. For more information about how to back up, restore, and modify the registry, see Microsoft Knowledge Base article 256986 Windows registry information for advanced users (http://go.microsoft.com/fwlink/?LinkId=62729).

Note

This setting is global and cannot be changed for individual application pools or applications.

1. Click **Start**, click **Run**, type **regedit.exe**, and then click **OK** to start Registry Editor.
2. Navigate to **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ASP.NET\4.0.30319.0**.
3. Create the key by following these steps:
   a. On the **Edit** menu, click **New**, and then click **Key**.
   b. Type **maxConcurrentRequestsPerCPU**, and then press **ENTER**.
   c. Under the **maxConcurrentRequestsPerCPU** key, create a DWORD entry with the new value for maxConcurrentRequestsPerCPU.
   d. Close Registry Editor.

To set the maxConcurrentRequestsPerCPU value for an application pool in the config section of an aspnet.config file

Note
Microsoft .NET Framework 4 must be installed to accommodate setting the values below via configuration file. You can download Microsoft .NET Framework 4 from Microsoft .NET Framework 4 (Web Installer) (http://go.microsoft.com/fwlink/?LinkID=189318).

- Open the aspnet.config file for the application pool, and then enter new values for the **maxConcurrentRequestsPerCPU** and **requestQueueLimit** parameters.

  The values in the following example are the default values.

```
<system.web>
    <applicationPool maxConcurrentRequestsPerCPU="5000"
requestQueueLimit="5000"/>
</system.web>
```

Note
This value overrides the value specified for **maxConcurrentRequestsPerCPU** in the registry. The **requestQueueLimit** setting is the same as processModel/requestQueueLimit, except that the setting in the aspnet.config file will override the setting in the machine.config file.

Define CLR hosting thread values for BizTalk host instances

Because a Windows thread is the most basic executable unit available to a Windows process, it is important to allocate enough threads to the .NET thread pool associated with an instance of a BizTalk host to prevent thread starvation. When thread starvation occurs, there are not enough

threads available to perform the requested work that negatively impacts performance. At the same time, care should be taken to prevent allocating more threads to the.NET thread pool associated with a host than is necessary. The allocation of too many threads to the .NET thread pool associated with a host may increase context switching. Context switching occurs when the Windows kernel switches from running one thread to a different thread, which incurs a performance cost. Excessive thread allocation can cause excessive context switching, which will negatively impact overall performance. The default number of threads allocated to a BizTalk host instance's .NET thread pool depends on which version of the .NET Framework is installed. The .NET Framework 4 and the .NET Framework 3.5 SP1 greatly increased the defaults, which can cause excessive thread allocation on the BizTalk Server computers and excessive lock contention on the MessageBox database.

Using the **BizTalk Settings Dashboard**, you can modify the default value for the number of Windows threads available in the .NET thread pool associated with an instance of a BizTalk host. For more information about how to modify .NET CLR settings, see How to Modify .NET CLR Settings (http://go.microsoft.com/fwlink/?LinkID=205344).

Note

Worker threads are used to handle queued work items and **I/O threads** are dedicated callback threads associated with an I/O completion port to handle a completed asynchronous I/O request.

Threading settings	Default value	Recommended value
Maximum IO Threads	250	250
Maximum Worker Threads	25	100 **Important** Increasing this value beyond 100 can have an adverse effect on the performance of the SQL Server computer hosting the BizTalk Server MessageBox database. When this problem occurs, SQL Server may encounter a deadlock condition. We recommend not increasing this parameter beyond a value of 100.
Minimum IO Threads	25	25

Threading settings	Default value	Recommended value
Minimum Worker Threads	5	25

Note
The recommended values are not absolutes and may need to be adjusted depending on the BizTalk application. Change one parameter at a time, and then measure the impact on the performance and stability of the BizTalk platform before making additional changes.

Note
These values are implicitly multiplied by the number of processors on the server. For example, setting the **MaxWorkerThreads** entry to a value of 100 would effectively set a value of 400 on a 4 CPU server.

Disable BizTalk Server Group-level tracking

Tracking incurs performance overhead within BizTalk Server as data has to be written to the MessageBox database and then asynchronously moved to the BizTalk Tracking database. The following considerations apply when configuring tracking in a production BizTalk Server environment:

- As a rule of thumb, if tracking is not a business requirement, then disable group-level tracking to reduce overhead and increase performance.

  To disable BizTalk Server group-level tracking, perform the following steps:

  a. In the **BizTalk Server Administration Console**, expand **BizTalk Server Administration**, right-click **BizTalk Group**, and then click **Settings**.
  b. In the BizTalk Settings Dashboard dialog box, on the Group page, clear the **Enable group-level tracking** check box.
  c. Click **OK** to apply the modifications and exit the Settings Dashboard.
- Only use message body tracking if necessary. Depending on message throughput and message size, message body tracking can cause significant overhead. While BizTalk activity tracking has obvious benefits for debugging and auditing, it also has considerable performance and scalability implications. Therefore, you should track only data that is strictly necessary for debugging and security reasons, and avoids tracking redundant information.
- By default, the following tracking options are enabled for orchestrations:
  - Orchestration start and end
  - Message send and receive
  - Shape start and end

  The orchestration tracking option “Shape start and end” incurs significant overhead and should not be enabled in a production environment where high throughput is necessary. Orchestration tracking options are accessible in the BizTalk Administration console on the **Tracking** page of the Orchestration Properties dialog box.

For more information about configuring tracking, see Configuring Tracking Using the BizTalk Server Administration Console (http://go.microsoft.com/fwlink/?LinkId=158021).

Decrease the purging period for the DTA Purge and Archive job from 7 days to 2 days in high throughput scenarios

By default, the purging interval for tracking data in BizTalk Server is set to 7 days. In a high throughput scenario, this can result in an excessive build up of data in the Tracking database, which will eventually impact the performance of the MessageBox and in turn negatively impact message processing throughput.

In high throughput scenarios, reduce the hard and soft purging interval from the default of 7 days to 2 days. For more information about configuring the purging interval, see How to Configure the DTA Purge and Archive Job (http://go.microsoft.com/fwlink/?LinkID=153814) in the BizTalk Server 2010 Help.

Configure the %temp% path for the BizTalk Service account to point to a separate disk or LUN

This should be done because BizTalk uses the temp location to stream files to disk when performing complex mapping operations.

Install the latest service packs

The latest service packs for both BizTalk Server and the .NET Framework should be installed, as these contain fixes that can correct performance issues you may encounter.

Performance optimizations in the BizTalk Server documentation

Apply the following recommendations from the BizTalk Server documentation as appropriate:

- Troubleshooting MessageBox Latency Issues (http://go.microsoft.com/fwlink/?LinkId=158019)
- Identifying Performance Bottlenecks (http://go.microsoft.com/fwlink/?LinkID=154238)
- Avoiding DBNETLIB Exceptions (http://go.microsoft.com/fwlink/?LinkID=155308)
- Avoiding TCP/IP Port Exhaustion (http://go.microsoft.com/fwlink/?LinkID=153240)
- Setting the EPM Threadpool Size (http://go.microsoft.com/fwlink/?LinkId=158020)

See Also

Optimizing BizTalk Server Performance

Low-Latency Scenario Optimizations

By default, BizTalk Server is optimized for throughput rather than low-latency. The following optimizations can be applied to BizTalk Server in scenarios where reduced latency is required.

Note
These optimizations will improve latency, but may do so at some cost to overall throughput.

Increase the BizTalk Server host internal message queue size

Each BizTalk host has its own internal in-memory queue. Increase the size of this queue from the default value of 100 to 10000 to improve performance for a low-latency scenario. For more information about modifying the value of the internal message queue size, see How to Modify Resource Based Throttling Settings (http://go.microsoft.com/fwlink/?LinkID=208366) in the BizTalk Server documentation.

Note
Increasing the internal message queue size value will increase the memory used by the host instance.

Increase the BizTalk Server host in-process messages

Increase the in-process messages from the default value of 1000 to 10,000 to improve performance. For more information about modifying the value of the in-process messages, see How to Modify the Default Host Throttling Settings (http://go.microsoft.com/fwlink/?LinkID=208366) in the BizTalk Server documentation.

Note
Increasing the internal message queue size value will increase the memory used by the host instance.

Optimize orchestrations for low latency

Follow the recommendations in the “Recommendations for optimizing orchestrations for low latency scenarios” section of Optimizing Orchestration Performance.

Configure polling intervals

Use the Settings Dashboard to configure the polling intervals of a given host, across the BizTalk Group. To change Polling Intervals:

1. In the **BizTalk Server Administration Console**, expand **BizTalk Server Administration**, right-click **BizTalk Group**, and then click **Settings**.
2. In the **BizTalk Settings Dashboard** dialog box, on the **Hosts** page, on the **General** tab, under **Polling Intervals**, you will find the **Messaging** and **Orchestration** values. By default, both these values are set to 500 milliseconds.

The following table lists the polling values that we used for testing on the BizTalk in-process 64-bit Hosts (RxHost, TxHost and PxHost). To disable polling, you can set the polling interval to a very big number as listed in the table.

Server Hosts	Messaging	Orchestration
RxHost Because we are only publishing incoming messages to the BizTalk message box through a one-way receive	200000	200000

Server Hosts	Messaging	Orchestration
location, polling is not required on the RxHost (receive host).		
TxHost Because we are only receiving messaging instances from the BizTalk message box, orchestration polling is not required on the TxHost (send host).	50	200000
PxHost Because we are only receiving orchestration instances from the BizTalk message box, messaging polling is not required on the PxHost (Processing host).	200000	50

See Also

Optimizing BizTalk Server Performance

Optimizing MQSeries Adapter Performance

Configure the MQSeries adapter by using the following settings when possible to increase performance.

Adjust the value for the MQSeries receive adapter polling threads

Increase the value specified for **Threads** in the **MQSeries Transport Properties** when configuring MQSeries adapter receive locations. Increasing this property from the default value of 2 to a value of 5 will significantly improve the rate at which messages can be received using the MQSeries adapter.

Disable transaction support on MQSeries receive locations where not required

MQSeries adapter transaction support incurs significant overhead. If transaction support is not a business requirement, set the **Transaction Supported** value in the **MQSeries Transport Properties** dialog box from the default value of **Yes** to **No**.

Disable Ordered delivery of messages on the MQSeries Adapter when not required

Enabling this property will enforce ordered delivery of messages as they are received from or sent to the MQSeries queue but will affect performance. When ordered delivery is enabled, the messaging runtime will use a single thread to receive messages from a given MQSeries queue. If

ordered delivery of messages is not a business requirement, then do not change the default value of **Ordered** property in the **MQSeries Transport Properties** dialog box which is set as **No Ordering**.

See Also

Optimizing BizTalk Server Performance

Optimizing Business Activity Monitoring (BAM) Performance

This topic describes Business Activity Monitoring (BAM) performance factors.

BAM disk usage configuration

BAM incurs significant overhead when a BizTalk system is under load because of the significant amount of data that is persisted to the BAM database. Therefore, judicious use of disk I/O techniques for the BAM database is critically important.

BAM EventStream APIs

Four types of EventStreams are available for use in a BizTalk BAM scenario:

- DirectEventStream (DES)
- BufferedEventStream (BES)
- OrchestrationEventStream (OES)
- MessageEventStream (MES)

You should choose one of these APIs based on the following factors:

- If your concern is latency, choose DES, where data is persisted synchronously to the BAM Primary Import database.
- If your concern is performance and throughput of event insertion, choose an asynchronous API (BES, OES or MES).
- If you are writing an application that runs on a computer that does not have BizTalk Server installed, use DES and BES; these APIs can be used in non-BizTalk applications.

  Note
  There are scenarios in which you may want to mix EventStream types. For example, for pipeline processing, you may want to capture the particular data in BAM regardless of whether the pipeline is rolling back its transaction. In particular, you may want capture data about how many messages failed or how many retries occurred during pipeline processing. To capture the data in this situation you should use BES.

- If your application runs on a computer on which BizTalk Server is installed, use MES and OES. (These APIs are available only from BizTalk applications.)

  Note
  OES is the equivalent of MES but for BizTalk orchestrations.

- If you want BAM event persistence to be in sync with pipeline transaction, you should use a Messaging Event Stream (MES).

All the asynchronous EventStreams (BES, MES, and OES) persist data first to the BizTalk MessageBox database. Periodically the data is processed and persisted to the BAM Primary Import database by the Tracking Data Decode Service (TDDS).

For more information about the BAM EventStream APIs, see EventStream Classes (http://go.microsoft.com/fwlink/?LinkId=158046) in the BizTalk Server documentation.

BAM performance counters

For a detailed list of the performance counters for BAM, see BAM Performance Counters (http://go.microsoft.com/fwlink/?LinkId=158048) in the BizTalk Server documentation.

See Also

Optimizing BizTalk Server Performance

Optimizing Business Rule Engine (BRE) Performance

The following factors should be considered when implementing the Business Rule Engine (BRE) in a BizTalk Server solution:

Fact types

The rule engine takes less time to access .NET facts compared to the time it takes to access XML and database facts. If you have a choice of using either .NET or XML or database facts in a policy, you should consider using .NET facts for improved performance.

Data table vs. data connection

When the size of the data set is small (< 10 or so), the **TypedDataTable** binding provides better performance than the **DataConnection** binding. However, the **DataConnection** binding performs better than the **TypedDataTable** binding when the data set is large (greater than or equal to 10 rows approximately). Therefore, you should decide whether to use the **DataConnection** binding or **TypedDataTable** binding based on the estimated size of the data set.

Fact retrievers

A fact retriever implements standard methods which are typically used to supply long-term and slowly changing facts to the rule engine before a policy is executed. The engine caches these facts and uses them over multiple execution cycles. Instead of submitting a static or fairly static fact each time that you invoke the rule engine, you should create a fact retriever that submits the fact for the first time, and then updates the fact in memory only when necessary.

Rule priority

The priority setting for a rule can range on either side of **0**, with larger numbers having higher priority. Actions are executed in order from the highest priority to lowest priority. When the policy implements forward-chaining behavior by using **Assert/Update** calls, the chaining can be optimized by using the priority setting. For example, assume that **Rule2** has a dependency on a value set by **Rule1**. Giving **Rule1** a higher priority means that **Rule2** will only execute after **Rule1** fires and updates the value. Conversely, if **Rule2** were given a higher priority, it could fire once,

and then fire again after **Rule1** fires and update the fact that **Rule2** is using a condition. While this may provide a correct result, giving **Rule1** a higher priority in this scenario will provide better performance.

Update calls

The Update function causes all the rules using the updated facts to be reevaluated. Update function calls can be expensive especially if a large set of rules is reevaluated when updating facts. There are situations where this behavior can be avoided. For example, consider the following rules.

Rule1:

```
IF PurchaseOrder.Amount > 5

THEN StatusObj.Flag = true; Update(StatusObj)
```

Rule2:

```
IF PurchaseOrder.Amount <= 5

THEN StatusObj.Flag = false; Update(StatusObj)
```

All remaining rules of the policy use **StatusObj.Flag** in their conditions. Therefore, when **Update** is called on the **StatusObj** object, all rules will be reevaluated. Whatever the value of the **Amount** field is, all rules except **Rule1** or **Rule2** are evaluated twice, once before the **Update** call and once after the **Update** call.

To mitigate the associated overhead, you could set the value of the **flag** field to **false** prior to invoking the policy and then use only **Rule1** in the policy to set the flag. In this case, **Update** would be called only if the value of the **Amount** field is greater than 5, and the **Update** function is not called if the value of **Amount** is less than or equal to 5. Therefore, all the rules except **Rule1** or **Rule2** are evaluated twice only if the value of the **Amount** field is greater than 5.

Usage of logical OR operators

Using an increasing number of logical OR operators in conditions creates additional permutations that expand the analysis network of the rule engine. From a performance standpoint, you are better off splitting the conditions into atomic rules that do not contain logical OR operators.

Caching settings

The Rule Engine uses two caches. The first one is used by the update service and the second one is used by each BizTalk process. The first time a policy is used, the BizTalk process requests the policy information from the update service. The update service retrieves the policy information from the rule engine database, caches it and returns the information to the BizTalk process. The BizTalk process creates a policy object based on that information and stores the policy object in a cache when the associated rule engine instance completes execution of the policy. When the same policy is invoked again, the BizTalk process reuses the policy object from the cache if one is available. Similarly, if the BizTalk process requests information about a policy from update service, the update service looks for the policy information in its cache if it is available. Every 60 seconds, the update service also checks if there have been any updates to the policy in the

database. If there are any updates, the update service retrieves the information and caches the updated information.

There are three tuning parameters for the rule engine related to these caches: **CacheEntries**, **CacheTimeout**, and **PollingInterval**. You can specify the values for these parameters either in the registry or in a configuration file. The value of the **CacheEntries** parameter is the maximum number of entries in the cache and is set to a value of 32 by default. You may want to increase the value of the **CacheEntries** parameter to improve performance in certain scenarios. For example, say you are using 40 policies repeatedly; you could to increase the value of the **CacheEntries** parameter to 40 to improve performance. This would allow the update service to maintain cache details of up to 40 policies in memory.

The value of **CacheTimeout** is the time in seconds that an entry is maintained in the update service cache. In other words, the **CacheTimeout** value refers to how long a cache entry for a policy is maintained in the cache without being referenced. The default value of **CacheTimeout** parameter is 3600 seconds, or 1 hour. It means that if the cache entry is not referenced within an hour, the entry is deleted. In some cases, it may be beneficial to increase the value of the **CacheTimeout** parameter to improve performance. For example, if a policy is invoked every two hours, performance of the policy execution would be improved by increasing the **CacheTimeout** parameter to a value higher than two hours.

The **PollingInterval** parameter of the rule engine defines the time in seconds for the update service to check the rule engine database for updates. The default value for the **PollingInterval** parameter is 60 seconds. If you know that the policies do not get updated at all or are updated rarely, you could change this parameter to a higher value to improve performance.

SideEffects property

The **ClassMemberBinding**, **DatabaseColumnBinding**, and **XmlDocumentFieldBinding** classes have a property named **SideEffects**. This property determines whether the value of the bound field, member, or column is cached. The default value of the **SideEffects** property in the **DatabaseColumnBinding** and **XmlDocumentFieldBinding** classes is **false**. The default value of the **SideEffects** property in the **ClassMemberBinding** class is **true**. Therefore, when a field of an XML document or a column of a database table is accessed for the second time or later within the policy, its value is retrieved from the cache. However, when a member of a .NET object is accessed for the second time or later, the value is retrieved from the .NET object, and not from the cache. Setting the **SideEffects** property of a .NET **ClassMemberBinding** to **false** will improve performance because the value of the field is retrieved from the cache from the second time onwards. You can only do this programmatically. The Business Rule Composer tool does not expose the **SideEffects** property.

Instances and selectivity

The **XmlDocumentBinding**, **ClassBinding**, and **DatabaseBinding** classes have two properties: **Instances** and **Selectivity**. The value of Instances is the expected number of instances of the class in working memory. The value of **Selectivity** is the percentage of the class instances that will successfully pass the rule conditions. The rule engine uses these values to optimize the condition evaluation so that the fewest possible instances are used in condition evaluations first

and then the remaining instances are used. If you have prior knowledge of the number of instances of the object, setting the **Instances** property to that value would improve performance. Similarly, if you have prior knowledge of the percentage of these objects passing the conditions, setting the **Selectivity** property to that value would improve performance. You can only set values for these parameters programmatically. The Business Rule Composer tool does not expose them.

See Also

Optimizing BizTalk Server Performance

Optimizing BizTalk Server WCF Adapter Performance

This topic provides recommendations for selecting the appropriate WCF adapter or binding type and guidance for applying various WCF adapter configuration options.

Considerations when choosing which WCF adapter to use or which WCF-Custom/WCF-CustomIsolated binding type to use

- Do not use message-level security if not strictly required because it increases the size of messages. This in turn can minimize the overall throughput of the solution.
- When choosing which WCF adapter to use or which WCF-Custom/WCF-CustomIsolated **binding type** to use, carefully consider any application requirements such as compatibility, performance, hosting platform, security, and supported transports. The guidelines listed below are applicable to WCF in general, and are not specific to BizTalk Server:
    - **BasicHttpBinding** should be used if the WCF service needs to support legacy clients such as WebSphere Web services or .NET 1.1 applications that expect an ASMX Web service. Because BasicHttpBinding does not implement any security by default, if you require message or transport security, you should configure it explicitly on this binding. Use BasicHttpBinding to expose endpoints that are able to communicate with ASMX-based Web services and clients and other services that conform to the WS-I Basic Profile 1.1. When configuring transport security, BasicHttpBinding defaults to use no credentials just like a classic ASMX Web service. BasicHttpBinding allows you to host the WCF service in IIS 7.5 or IIS 7.0.
    - **WsHttpBinding** should be used if the WCF service will be called by WCF clients over the Internet. WsHttpBinding is a good choice for Internet scenarios in which you do not have to support legacy clients that expect an ASMX Web service and WsHttpBinding allows you to host the WCF service in IIS 7.5 or IIS 7.0. If you do need to support legacy clients, consider using BasicHttpBinding instead.
      WsHttpBinding should be used when you need to expose WCF Receive Locations or adopt Send Ports which support WS-* Protocols like WS-Security or WS-AtomicTransactions.
    - **NetTcpBinding** is an excellent choice if you need to support clients within your intranet. NetTcpBinding is a good choice for intranet scenarios if transport performance is important to you, and if it is acceptable to host the service in a Windows service instead of in IIS. Use this binding when you want to provide a secure and reliable binding

environment for .NET-to-.NET cross-machine communication. Note that the NetTcpBinding must be hosted in a Windows service or in IIS 7.5/7.0.

- **NetNamedPipeBinding** should be used if you need to support WCF clients on the same computer as your service. This binding provides a secure and reliable binding environment for cross-process, same-computer communication. Use this binding when you want to make use of the NamedPipe protocol. Note that the NetNamedPipeBinding must be hosted in a Windows service or in IIS 7.5/7.0.
- **NetMsmqBinding** should be used if you need to support disconnected queuing. Queuing is provided by using Message Queuing (also known as MSMQ) as a transport, which enables support for disconnected operations, failure isolation, and load leveling. You can use NetMsmqBinding when the client and the service do not have to be online at the same time. You can also manage any number of incoming messages by using load leveling. Message Queuing supports failure isolation, where messages can fail without affecting the processing of other messages. Note that the NetMsmqBinding must be hosted in a Windows service or in IIS 7.5/7.0.
- **WsDualHttpBinding** should be used if you need to support a duplex service. A duplex service is a service that uses duplex message patterns, thus providing the ability for a service to communicate back to the client via a callback. Note that the WsDualHttpBinding must be hosted in a Windows service or in IIS 7.5/7.0.

WCF Binding Comparison

Bindings provide a mechanism for configuring channel stacks. A binding creates a process to build a channel stack using a transport channel, a message encoder, and a set of protocol channels. Windows Communication Foundation ships with numerous built-in bindings that come preconfigured to address most common communication scenarios.

Binding Class Name	Transport	Message Encoding	Security Mode	Reliable Messaging	Transaction Flow (disabled by default)
BasicHttpBinding	HTTP	Text	None	Not supported	Not supported
WSHttpBinding	HTTP	Text	Message	Disabled	WS-AtomicTransactions
NetTcpBinding	TCP	Binary	Transport	Disabled	OleTransactions
NetNamedPipesBinding	Named Pipes	Binary	Transport	Not supported	OleTransactions
NetMsmqBinding	MSMQ	Binary	Message	Not supported	Not supported
CustomBinding	You decide	You decide	You decide	You decide	You decide

You can select a particular binding based on your communication needs. For example:

- **BasicHttpBinding** is designed for interoperability with simple Web services. BasicHttpBinding uses HTTP for the transport and text for the message encoding.
- **WSHttpBinding** is designed for interoperability with more advanced Web services that might take advantage of different WS-* protocols. WSHttpBinding also uses HTTP for the transport and text for the message encoding.
- **NetTcpBinding** and **NetNamedPipeBinding** are designed for efficient and perform ant communication with other WCF applications across machines or on the same machine respectively.
- If you need maximum flexibility by using one or multiple custom protocol channels at runtime, you can use the **CustomBinding** that gives you the possibility to decide which binding elements compose your binding.

Note

Bindings have different characteristics in terms of response time and throughput. So, the general advice to increase performance is using the NetTcpBindind and NetNamesPipeBinding when possible.

Use the TCP transport and binary message encoding to maximize WCF adapter throughput and minimize WCF adapter latency

Use the WCF-NetTcp adapter when possible to maximize throughput. The WCF-NetTcp adapter uses TCP/IP transport protocol and binary message encoding which together provide improved performance as compared to other WCF-* adapters.

Alternatively, you can configure the WCF-Custom (or WCF-CustomIsolated adapter for Receive Locations) with a **customBinding** binding type. Then, add the **binaryMessageEncoding** binding extension to implement binary message encoding and the **tcpTransport** binding extension to implement TCP/IP as the message transport protocol. This approach is very flexible because it allows you to select and configure only the binding elements you need and to create custom channels to extend the default behavior of the BizTalk Messaging Engine. If you implement the WCF-Custom adapter with the **customBinding** binding type, specify these values for the binding extension Configuration parameters to maximize throughput and minimize latency.

Send Port Configuration Values:

Setting	Default value	Recommended value
TcpTransportBindingElement.ConnectionPoolSettings.maxOutboundConnectionsPerEndpoint - Gets or sets the maximum number of outbound connections for each endpoint cached in the connection pool. This limits the number of connections that are cached for each unique remote endpoint. If this value is exceeded by having more active client connections, the service may appear unresponsive to the client, and this value should be adjusted to accommodate the maximum number of expected connections that are cached for each unique remote endpoint. For more information about this	10	Try >= 20

Setting	Default value	Recommended value
property, see TcpConnectionPoolSettings.MaxOutboundConnectionsPerEndpoint Property (http://go.microsoft.com/fwlink/?LinkId=157737) on MSDN.		

Receive Port Configuration Values:

Setting	Default value for .NET Framework 4	Recommended value for .NET Framework 4	Default value for .NET Framework 3.5	Recommended value for .NET Framework 3.5
TcpTransportBindingElement.MaxPendingAccepts - Gets or sets the maximum number of pending asynchronous accept operations that are available for processing incoming connections to the service. For scenarios with high levels of simultaneous connection initiations, this value may be inadequate and should be adjusted along with the **MaxPendingConnections** property to handle higher levels of concurrent client connection attempts. It should not be necessary to set this property to a value greater than the number of processors present in the machine hosting the service. For more information about this property, see ConnectionOrientedTransportBindingElement.MaxPendingAccepts Property (http://go.microsoft.com/fwlink/?LinkId=157738) on MSDN.	1	2*Processor Count	1	Try >= 20
TcpTransportBindingElement.MaxPendingConnections - Gets or sets the maximum number of connections awaiting dispatch on the service. This limits the number of simultaneous client connections awaiting dispatch. If this value is too	10	16*Processor Count	10	Try >= 20

Setting	Default value for .NET Framework 4	Recommended value for .NET Framework 4	Default value for .NET Framework 3.5	Recommended value for .NET Framework 3.5
low, client connection attempts may be rejected by the service. If it is too high, the service may appear slow or unresponsive to clients during heavy load periods. This property should be set to a value that allows the service to run at full capacity, and no higher. When a higher layer in the stack calls **AcceptDispatch**, that connection is removed from the queue of connections awaiting dispatch. For more information about this property, see ConnectionOrientedTransportBindingElement.MaxPendingConnections Property (http://go.microsoft.com/fwlink/?LinkId=157735) on MSDN.				
TcpTransportBindingElement.ListenBacklog - Gets or sets the maximum number of queued connection requests that can be pending. **ListenBacklog** is a socket-level property that describes the number of "pending accept" requests to be queued. Ensure that the underlying socket queue is not exceeded by the maximum number of concurrent connections. For more information about this property see TcpTransportBindingElement.ListenBacklog Property (http://go.microsoft.com/fwlink/?LinkId=157734) on MSDN.	10	16*Processor Count	10	Try >= 20

Add the ServiceThrottlingBehavior service behavior to a WCF-Custom or WCF-CustomIsolated Receive Location and use the following settings:

Note
Before any elements of the ServiceThrottlingBehavior service behavior can be modified, you must first add the **serviceThrottling** behavior extension to the **Behaviors** tab of the

WCF-Custom* Transport Properties dialog box. To add serviceThrottling to the list of Behaviors, select the **Behaviors** tab of the **WCF-Custom* Transport Properties** dialog box, right-click **ServiceBehavior** under **Behavior**, click **Add extension**, select **serviceThrottling**, and then click **OK**. Then click to select the properties available under **ServiceThrottlingElement** and change the value for the properties as needed.

Setting	Default value for .NET Framework 4	Recommended value for .NET Framework 4	Default value for .NET Framework 3.5	Recommended value for .Net Framework 3.5
ServiceThrottlingBehavior.MaxConcurrentCalls - Gets or sets a value that specifies the maximum number of messages actively processing across a **ServiceHost**. The **MaxConcurrentCalls** property specifies the maximum number of messages actively processing across a **ServiceHost** object. Each channel can have one pending message that does not count against the value of **MaxConcurrentCalls** until WCF begins to process it. For more information about this property, see ServiceThrottlingBehavior.MaxConcurrentCalls (http://go.microsoft.com/fwlink/?LinkId=157838) on MSDN. The default value for the BizTalk WCF-Custom and WCF-CustomIsolated receive adapters **MaxConcurrentCalls** property is **16** per CPU. **Note** BizTalk Server WCF receive adapters other than the WCF-Custom and WCF-CustomIsolated adapter expose a **Maximum Concurrent Calls** property on the **Binding** tab of the **WCF-* Transport Properties** dialog box to configure this behavior. The default value of the **Maximum Concurrent Calls** behavior is **200**.	16*ProcessorCount	16*ProcessorCount	• 16 for the BizTalk WCF-Custom and WCF-CustomIsolated receive adapters. • 200 for the other WCF receive adapters.	• Try >= 200 for the WCF-Custom and WCF-CustomIsolated receive adapters. • Try > 200 for the **Maximum Concurrent Calls** prop

Setting	Default value for .NET Framework 4	Recommended value for .NET Framework 4	Default value for .NET Framework 3.5	Recommended value for .Net Framework 3.5
				erty on the **Binding** tab of the **WCF-* Transport Properties** dialog box for BizTalk Server WCF receive adapters other than the WCF-Custom and WCF-

Setting	Default value for .NET Framework 4	Recommended value for .NET Framework 4	Default value for .NET Framework 3.5	Recommended value for .Net Framework 3.5
				CustomIsolated adapter.
ServiceThrottlingBehavior.maxConcurrentInstances - Gets or sets a value that specifies the maximum number of **InstanceContext** objects in the service that can execute at once. The **MaxConcurrentInstances** property specifies the maximum number of **InstanceContext** objects in the service. It is important to keep in mind the relationship between the **MaxConcurrentInstances** property and the **InstanceContextMode** property. If **InstanceContextMode** is PerSession, the resulting value is the total number of sessions. If **InstanceContextMode** is PerCall, the resulting value is the number of concurrent calls. If a message arrives while the maximum number of **InstanceContext** objects already exist, the message is held until an **InstanceContext** object closes. For more information about this property, see ServiceThrottlingBehavior.MaxConcurrentInstances Property (http://go.microsoft.com/fwlink/?LinkId=157897) on MSDN. The default value for the WCF-Custom and WCF-CustomIsolated receive adapter MaxConcurrentInstances property is **116** per CPU. **Note** Since WCF receive locations are implemented as instances of the BizTalkServiceInstance class	116*ProcessorCount	116*ProcessorCount	26	Try >= 200

Setting	Default value for .NET Framework 4	Recommended value for .NET Framework 4	Default value for .NET Framework 3.5	Recommended value for .Net Framework 3.5
contained in the Microsoft.BizTalk.Adapter.Wcf.Runtime.dll assembly and since this class is marked as ServiceBehavior(InstanceContextMode=InstanceContextMode.Single, ConcurrencyMode=ConcurrencyMode.Multiple). all incoming requests are managed by the same singleton object and this parameter is ignored. So setting the maxConcurrentInstances property on certain WCF-Custom receive locations has no effect, since the number of active instances is always equal to 1.				
ServiceThrottlingBehavior.MaxConcurrentSessions - The **MaxConcurrentSessions** property gets or sets the maximum number of sessions a **ServiceHost** object can accept at one time. It is important to understand that sessions in this case does is not limited to only channels that support reliable sessions. Each listener object can have one pending channel session that does not count against the value of **MaxConcurrentSessions** until WCF accepts the channel session and begins processing the channel session messages. This property is most useful in scenarios that make use of sessions. When this property is set to a value less than the number of client threads, the requests from multiple clients may get queued in the same socket connection. The requests from the client that has not created a session with the service will be blocked. They will remain blocked until the	100*ProcessorCount	100*ProcessorCount	10	Try >= 200

Setting	Default value for .NET Framework 4	Recommended value for .NET Framework 4	Default value for .NET Framework 3.5	Recommended value for .Net Framework 3.5
service closes its session with the other clients, if the number of open sessions on the service has reached the value specified for **MaxConcurrentSessions**. The client requests that are not served are then timed out, and the service closes the session. To avoid this situation, consider running the client threads from different application domains so that the request messages are accepted by different socket connections. For more information about this property, see ServiceThrottlingBehavior.MaxConcurrentSessions Property (http://go.microsoft.com/fwlink/?LinkId=157864) on MSDN.				

When modifying port configuration settings apply changes methodically; modify each parameter individually and, test the effects of the change on performance and overall stability. As always, the proper value to apply depends on the particular scenario: if a value is set too low, scalability can be reduced; whereas if a value is set too high, the application requirement may exceed the capacity of physical resources on the computer. In addition, since these properties are related, they should be set in a consistent and coherent way. For more information about using ServiceThrottlingBehavior to control WCF Service Performance, see Using ServiceThrottlingBehavior to Control WCF Service Performance (http://go.microsoft.com/fwlink/?LinkId=157908) on MSDN.

See Also

Optimizing BizTalk Server Performance

Optimizing Orchestration Performance

This topic describes best practices for using orchestrations in BizTalk Server solutions. This includes recommendations for:

- Reducing latency of BizTalk Server solutions that use orchestrations
  - Eliminating orchestrations for messaging only patterns
  - Utilizing inline sends from orchestrations

    - Minimizing orchestration persistence points
- Nesting orchestrations
- Orchestration design patterns
- Orchestration exception handling blocks

Recommendations for optimizing orchestrations for low latency scenarios

The following techniques can be used to reduce latency of BizTalk Server solutions that use orchestrations.

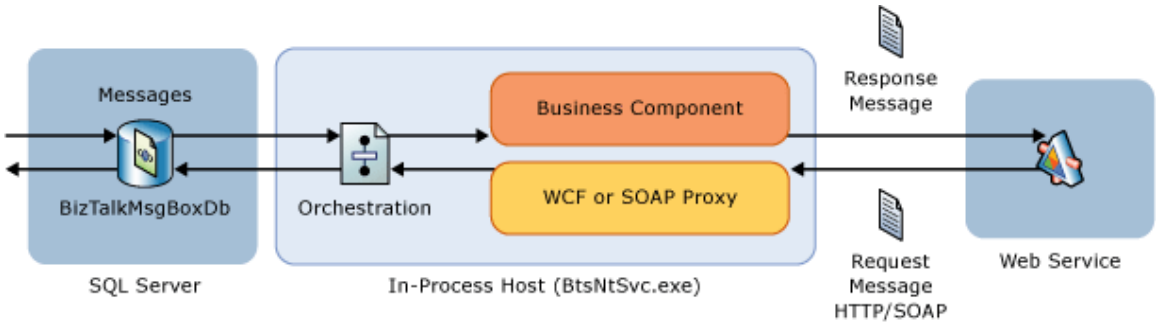
Eliminate orchestrations for messaging only patterns

When possible minimize the use of orchestrations to increase overall throughput and reduce the latency of business processes. If there is no need to run long running transactions and there is no need to invoke several systems for each request, then consider eliminating orchestrations and moving business logic to Receive and Send Ports to reduce the total amount of roundtrips to the BizTalkMsgBoxDb and decrease the latency due to database access. In this case, implement custom pipelines and reuse helper classes that were previously invoked from orchestrations. Use orchestrations only when strictly needed to implement design patterns as Scatter and Gather or Convoys. For more information about Orchestration Design Patterns, see the topic Implementing Design Patterns in Orchestrations (http://go.microsoft.com/fwlink/?LinkId=140042) in the BizTalk Server documentation.

Use inline sends from orchestrations to accommodate low latency scenarios

Whenever possible, minimize the use of orchestrations and favor messaging-only patterns to increase the overall throughput and reduce the latency of the business processes. If there is no need for long-running transactions and there is no need for business logic to invoke several systems, then consider moving business logic to receive and send Ports and eliminating the use of orchestrations. This approach can be implemented with custom pipelines and which reuse the helper classes that were previously invoked from orchestrations. In order to achieve better performance in low latency scenarios, adopt one of the following approaches:

- Eliminate unnecessary orchestrations and adopt messaging-only patterns to reduce the total amount of roundtrips to the BizTalk MessageBox database. This approach accommodates low latency because inline sends bypass the BizTalk messaging engine and the associated overhead. Orchestration inline send functionality is provided with BizTalk Server 2006 and later.
- Inside orchestrations, eliminate logical ports bound to physical ports and use in-line sends in their place. For example, an inline send could be used to create an instance of a WCF proxy class to invoke a downstream Web service or an ADO.NET component to access a SQL Server database. In the following diagram, an inline send is performed when the orchestration instantiates a business component, which internally makes use of a WCF proxy object to directly invoke a downstream Web service:



Note

Although using inline sends from orchestrations will significantly reduce latency, there are limitations to this approach. Because inline sends bypass the BizTalk messaging engine, the following functionality provided with the messaging engine is not available:

- Transactional pipelines
- Recoverable pipelines
- Pipelines that call the BAM interceptor API
- BizTalk Server adapter support
- Batching
- Retries
- Correlation set initialization
- Declarative configuration
- Secondary transports
- Tracking
- Declarative use of BAM

For more information about the types of pipelines that cannot be executed from within an orchestration, see the “Restrictions” section of the topic How to Use Expressions to Execute Pipelines (http://go.microsoft.com/fwlink/?LinkId=158008) in the BizTalk Server 2010 documentation.

Optimize orchestration latency by reducing the number of persistence points, if possible

An orchestration scope only needs to be marked as “long-running transactional” if you want to use compensation or scope timeouts. A long-running transactional scope causes an extra persistence point at the end of the scope, so they should be avoided when you don’t need to use compensation or scope timeouts. An atomic scope causes a persistence point at the end of the scope, but also guarantees that no persistence points will occur inside the atomic scope. This side-effect of no persistence inside the scope can sometimes be used to your advantage to batch persistence points (when doing multiple sends, for example). In general, though, we recommend avoiding atomic scopes if possible. The orchestration engine saves to persistent storage the entire state of a running orchestration instance at various points, so that the instance can later be restored in memory and carried out to completion.

The number of persistence points in an orchestration is one of the key factors influencing

the latency of messages flowing through orchestrations. Each time the engine hits a persistence point, the internal state of a running orchestration is serialized and saved to the MessageBox and this operation incurs a latency cost. The serialization of the internal state includes all messages and variables instantiated and not yet released in the orchestration. The larger the messages and variables and the greater the number of these, the longer it will take to persist the internal state of an orchestration. An excessive number of persistence points can lead to significant performance degradation. For this reason, we recommend eliminating unnecessary persistence points from orchestrations by reducing the number of transactional scopes and Send shapes. This approach allows decreasing the contention on the MessageBox due to orchestration dehydration and rehydration, increasing the overall scalability, and reducing orchestration latency. Another recommendation is to always keep the internal state of an orchestration as small as possible. This technique can significantly reduce the time spent by the XLANG Engine serializing, persisting and restoring the internal state of an orchestration in case of persistence point. One way to achieve this is to create variables and messages as late as possible and release them as early as possible; for example introduce non transactional scopes inside your orchestrations and declare variables and messages within these inner scopes instead of declaring them at the top-most level. For more information about minimizing orchestration persistence points, see the following topics in the BizTalk Server 2010 documentation:

- Persistence and the Orchestration Engine (http://go.microsoft.com/fwlink/?LinkID=155292).
- Orchestration Dehydration and Rehydration (http://go.microsoft.com/fwlink/?LinkID=155292).

Guidelines for using promoted properties to access message tags or attributes from an orchestration

If you do need to promote properties, promote only those properties that are used for message routing, filters, and message correlation. The promotion of each property requires the disassembler component (XML, Flat, custom) to recognize the document type and to retrieve data from the message using the XPath expression from the relative annotation contained in the XSD that defines the document type. In addition, each property promotion causes a separate call of the bts_InsertProperty stored procedure when the Message Agent publishes the message to the MessageBox database. If an orchestration needs to access a particular element or attribute contained by an XML document, use one of the following techniques:

- Reduce the number of written and promoted properties and eliminate those that are not strictly necessary.
- Eliminate unnecessary distinguished fields. The distinguished fields are written context properties and they can easily occupy significant space as their name is equal to the XPath expression that is used to retrieve data. The distinguished property is defined as annotations in the XSD that defines the document type. The disassembler component uses the same approach adopted for promoted properties and uses the XPath expression that defines a distinguished field to find it within in the incoming document. Then, the disassembler component writes a property in the context where:
    - **Name**: XPath Expression defined in the annotation.
    - **Value**: Value of the element identified by the XPath expression within an incoming document.

The XPath Expressions can be very long, especially when the element in question is very deep in the document schema. So, the more distinguished fields, the larger the context size. This in turn adversely affects the overall performance.

- Use the XPath built-in function provided by the orchestration runtime.
- If messages are quite small (a few kilobytes) and XML-formatted, you can de-serialize the message into a .NET class instance and work with public fields and properties. If the message needs a complex elaboration (custom code, Business Rule Engine policies, etc.) accessing data using the properties exposed by an instance of a .NET class is much faster using XPath expressions. When the business logic invoked by the orchestration has completed, the entity object can be serialized back into a BizTalk message. You can create .NET classes from an XML schema using one of the following tools: XSD tool (.NET Framework 2.0) or SVCUTIL (.NET Framework 3.0).
- Enable a helper component from an orchestration. This technique has an advantage over using distinguished fields. In fact, an orchestration may read the XPath expression from a config store (config file, SSO Config Store, custom Db, and so on) so, when you have to change the XPath expression, you do not have to change and redeploy a schema, as you should do for promoted properties and distinguished fields. The following code sample provides an example of a helper component. The component uses the XPathReader class that is provided by the BizTalk runtime. This allows the code to read through the document stream until the XPath expression is found.

```
#region Copyright

//=======================================================================================

//Microsoft Windows Server AppFabric Customer Advisory Team (CAT)

//

// This sample is supplemental to the technical guidance published on the community

// blog.

//

// Author: Paolo Salvatori.

//=======================================================================================

// Copyright © 2010 Microsoft Corporation. All rights reserved.

//

// THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER

// EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF

// MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. YOU BEAR THE RISK OF USING IT.

//=======================================================================================

#endregion

#region Using Directives

using System;
```

```
using System.Collections.Generic;
using System.IO;
using System.Xml;
using System.Linq;
using System.Text;
using System.Globalization;
using Microsoft.XLANGs.BaseTypes;
using Microsoft.BizTalk.Streaming;
using Microsoft.BizTalk.XPath;
#endregion
namespace Microsoft.AppFabric.CAT.Samples.DuplexMEP.Helpers
{
public class XPathHelper
{
#region Private Constants
private const string MessageCannotBeNull = "[XPathReader] The message cannot be null.";
#endregion
#region Public Static Methods
public static string GetValue(XLANGMessage message, int partIndex, string xpath)
{
try
{
if (message == null)
{
throw new ApplicationException(MessageCannotBeNull);
}
using (Stream stream = message[partIndex].RetrieveAs(typeof(Stream)) as Stream)
{
XmlTextReader xmlTextReader = new XmlTextReader(stream);
XPathCollection xPathCollection = new XPathCollection();
XPathReader xPathReader = new XPathReader(xmlTextReader, xPathCollection);
xPathCollection.Add(xpath);
while (xPathReader.Read())
{
```

```
if (xPathReader.HasAttributes)
{
for (int i = 0; i < xPathReader.AttributeCount; i++)
{
xPathReader.MoveToAttribute(i);
if (xPathReader.Match(xPathCollection[0]))
{
return xPathReader.GetAttribute(i);
}
}
}
if (xPathReader.Match(xPathCollection[0]))
{
return xPathReader.ReadString();
}
}
}
}
finally
{
message.Dispose();
}
return string.Empty;
}
#endregion
}
}
```

Minimize orchestration complexity to improve performance

The complexity of orchestrations has a significant impact on performance. As orchestration complexity increases, overall performance decreases. Orchestrations can be used in an almost infinite variety of scenarios, and each scenario might involve orchestrations of varying complexity. Avoid complex orchestrations when possible in favor of a modular approach. In other words, split your business logic into multiple, reusable orchestrations.

If you do need to implement a complex orchestration, define messages and variables into inner scopes whenever possible rather than at the root level. This technique maintains a smaller footprint in memory for each orchestration because variables and messages are disposed of when the flow leaves the scope where the variables and messages were defined. This approach is particularly beneficial when orchestrations are saved to the MessageBox at persistence points.

Use the Call Orchestration shape versus the Start Orchestration shape to improve performance

Avoid the **Start Orchestration** shape and use the **Call Orchestration** shape to execute a nested orchestration. In fact, The **Call Orchestration** shape can be used to synchronously call an orchestration that is referenced in another project. This approach allows for reuse of common orchestration workflow patterns across BizTalk projects. When you invoke another nested orchestration synchronously with the **Call Orchestration** shape, the enclosing orchestration waits for the nested orchestration to finish before continuing. The nested orchestration is executed on the same thread that runs the calling orchestration.

The **Start Orchestration** shape is similar to the **Call Orchestration** shape, but in this case the nested orchestration is called in an asynchronous manner: the flow of control in the invoking orchestration proceeds beyond the invocation, without waiting for the invoked orchestration to finish its work. In order to implement this decoupling between the caller and the called orchestrations, the **Start Orchestration** is implemented via publication of a message to the BizTalk Messagebox. This message is then consumed by an in-process BizTalk host instance which executes the nested orchestration. When possible, use **Call Orchestration**, especially if the calling orchestration needs to wait for a result from the nested orchestration in order to continue processing. For more information about using the Call Orchestration shape, see the following topics in the BizTalk Server 2010 documentation:

- Working with Direct Bound Ports in Orchestrations (http://go.microsoft.com/fwlink/?LinkId=139902).
- Nesting Orchestrations (http://go.microsoft.com/fwlink/?LinkId=139903).

Use XmlReader with XLANGMessage versus using XmlReader with XmlDocument to improve orchestration performance

To improve orchestration performance for .NET methods called from an orchestration, use XmlReader with XLANGMessage, not XmlDocument. The following code example illustrates this functionality.

```
// As a general rule, use XmlReader with XLANGMessage, not XmlDocument.

// This is illustrated in the parameter passed into the following code.

// The XLANG/s compiler doesn't allow a return value of XmlReader

// so documents must be initially constructed as XmlDocument()

public static XmlDocument FromMsg(XLANGMessage old)

{
```

```
    //get at the data
    XmlDocument ret = new XmlDocument();

    try{
        XmlReader reader = (XmlReader)old[0].RetrieveAs(typeof(XmlReader));
        //construct new message from old
        //read property
        object msgid = old.GetPropertyValue(typeof(BTS.MessageID));
    }
    finally {
        // Call Dispose on the XLANGMessage object
        // because the message doesn't belong to the
        // .NET runtime - it belongs to the Messagebox database
        old.Dispose();
    }
    return ret;
}
```

Another method would be to create a .NET class based on the schema. This takes less memory than loading the document into an **XmlDocument** object, as well as providing easy access to the schema elements for .NET developers. To generate a class based on a BizTalk schema, you can use the xsd.exe tool provided with Visual Studio. For example, running **xsd.exe <schema.xsd> /classes** against a simple schema containing fields named ItemA, ItemB, ItemC, will produce the following class.

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:2.0.50727.1433
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------


using System.Xml.Serialization;
```

```
//
// This source code was auto-generated by xsd, Version=2.0.50727.42.
//


/// <remarks/>
[System.CodeDom.Compiler.GeneratedCodeAttribute("xsd", "2.0.50727.42")]
[System.SerializableAttribute()]
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Xml.Serialization.XmlTypeAttribute(AnonymousType=true,
Namespace="http://Schemas.MySchema")]
[System.Xml.Serialization.XmlRootAttribute(Namespace="http://Schemas.MySchema",
IsNullable=false)]
public partial class MySchemaRoot {

    private string itemAField;

    private string itemBField;

    private string itemCField;

    /// <remarks/>
[System.Xml.Serialization.XmlElementAttribute(Form=System.Xml.Schema.XmlSchemaForm.Unqual
ified)]
    public string ItemA {
        get {
            return this.itemAField;
        }
        set {
            this.itemAField = value;
        }
    }
```

```
    /// <remarks/>

[System.Xml.Serialization.XmlElementAttribute(Form=System.Xml.Schema.XmlSchemaForm.Unqual
ified)]
    public string ItemB {
        get {
            return this.itemBField;
        }
        set {
            this.itemBField = value;
        }
    }

    /// <remarks/>

[System.Xml.Serialization.XmlElementAttribute(Form=System.Xml.Schema.XmlSchemaForm.Unqual
ified)]
    public string ItemC {
        get {
            return this.itemCField;
        }
        set {
            this.itemCField = value;
        }
    }
}
```

This class can then be referenced in your .NET assembly in order to access the message elements, and the returned object can be directly assigned to a message. The following is an example use of the class generated above.

```
public static Root SetValues(Microsoft.XLANGs.BaseTypes.XLANGMessage msg)
{
   try
   {
   MySchemaRoot rootObj=(MySchemaRoot)msg[0].RetrieveAs(typeof(MySchemaRoot);
```

```
   rootObj.ItemA="value a";
   rootObj.ItemB="value b";
   rootObj.ItemC="value c";
   }
    finally {
        msg.Dispose();
            }

   return rootObj;
}
```

This technique allows you to use an object-oriented approach when processing messages. This technique should be used primarily with relatively small messages. This is because even though this technique uses considerably less memory than when loading the message into an **XmlDocument** object, the entire message is still loaded into memory. When processing larger messages, use the **XmlReader** class to read messages and the **XmlWriter** class to write messages. When using **XmlReader** and **XmlWriter**, the message is contained in a **VirtualStream** object. If the message size exceeds the value specified for **Large message threshold (bytes)** that is exposed on the BizTalk Group Properties configuration page, the message is written to the file system. This decreases overall performance, but avoids out of memory exceptions.

Improve performance by minimizing the use of logical ports bound to physical ports

You can increase performance by minimizing the use of logical ports bound to physical ports that use the following adapters:

- SQL Server, Oracle
- WSE, HTTP, WCF
- MSMQ, MQSeries

In BizTalk Server 2010, send and receive pipelines can be directly invoked from an orchestration using the XLANGPipelineManager class contained in the Microsoft.XLANGs.Pipeline.dll. Thus, the processing of pipelines can be moved from ports to orchestrations; logical ports in an orchestration can be substituted with an Expression shape, which invokes an instance of a given .NET class (for example, a Data Access component using ADO.NET). Before adopting this technique, you should be aware that if you do not use adapters and physical ports, you lose the ability to leverage their functions, such as batching, retries, declarative configuration, and secondary transports.

Orchestration Design Patterns

The Orchestration Designer allows developers to implement a wide range of enterprise integration patterns. Some common patterns include Aggregator, Exception Handling and Compensation, Message Broker, Scatter and Gather, and Sequential and Parallel Convoy. Those patterns can be utilized to develop complex Enterprise Application Integration (EAI), Service-Oriented Architecture (SOA), and Business Process Management (BPM) solutions with BizTalk Server. For more information about Orchestration Design Patterns, see the topic Implementing Design Patterns in Orchestrations (http://go.microsoft.com/fwlink/?LinkId=140042) in the BizTalk Server documentation and Patterns and Best Practices for Enterprise Integration (http://go.microsoft.com/fwlink/?LinkId=140043).

Make appropriate use of .NET classes in orchestrations to maximize performance

In general, the .NET classes used inside an orchestration can be divided into two distinct categories:

- **Helpers and services -** These classes provide common services to orchestrations such as tracing, error handling, caching, and serialization/deserialization. Most of these classes can be implemented as static classes with no internal state and multiple public static methods. This approach avoids creating multiple objects of the same class in different orchestrations running at the same time, which helps to reduce the working space of host processes and save memory. A class that is stateless helps to reduce the overall size of the internal state that must be serialized and persisted to the BizTalk MessageBox when an orchestration is dehydrated.
- **Entities and Business Objects -** You can use these classes to manage entities, such as orders, order items, and customers. A single orchestration can internally create and manage multiple instances of the same type. These classes are typically stateful, and expose public fields and/or properties along with methods to modify the internal state of the object. Instances of these classes can be dynamically created by deserializing an XLANGMessage part into a .NET object by using the **XmlSerializer** or the **DataContractSerializer** classes or by using the **XLANGPart.RetrieveAs** method. You should structure an orchestration using non-transactional scopes in such a way that instances of stateful classes are created as late as possible and released as soon as they are no longer needed. This approach reduces the working space of host processes and minimizes the overall size of the internal state that is serialized and persisted to the MessageBox database when an orchestration is dehydrated. For more information about using orchestrations in BizTalk Server, see the article FAQ for BizTalk Server Orchestrations (http://go.microsoft.com/fwlink/?LinkID=116886).

  Note

  While this article is written for BizTalk Server 2004 and BizTalk Server 2006, the concepts presented also apply to BizTalk Server 2010 orchestrations.

Orchestration Exception Handler Blocks

When using Orchestration exception Handler blocks, include all orchestration shapes in one or multiple scopes (non transactional scopes whenever possible) and create at least the following exception handler blocks:

- An exception handler block for handling a generic System.Exception error.
- An exception handler block for handling a general exception in order to catch and handle possible unmanaged errors, such as COM exceptions.

For more information about using Orchestration exception handling blocks, see the following articles:

- Using BizTalk Server Exception Handling (http://msdn.microsoft.com/en-us/library/aa561229.aspx).
- Charles Young Blog, BizTalk Server 2006: The Compensation Model (http://go.microsoft.com/fwlink/?LinkId=158017).

Note

While this blog was written with BizTalk Server 2006 in mind, the principles described in the blog also apply to BizTalk Server 2010.

Considerations when using maps in orchestrations

The following considerations apply when using maps in orchestrations:

- If you are using a map to extract or set properties used with business logic in an orchestration, use distinguished fields or promoted properties. This practice should be followed because when extracting or setting values with a map the document is loaded into memory however when using distinguished fields or promoted properties, the orchestration engine accesses the message context and does not load the document into memory.
- If you are using a map to aggregate several fields into one field, use distinguished fields or promoted properties with an orchestration variable to accumulate the result set.

See Also

Optimizing Performance

Optimizing WCF Web Service Performance

WCF services expose numerous configuration parameters that affect performance. This topic provides general guidance for setting optimal values for these configuration parameters to improve performance of WCF services.

Implement serviceThrottling behavior for backend WCF services

Implement serviceThrottling behavior for backend WCF services. Service throttling allows you to even out the load on your backend WCF servers and to enforce resource allocation. serviceThrottling behavior for backend WCF services is configured by modifying the values for the **maxConcurrentCalls**, **maxConcurrentSessions**, and **maxConcurrentInstances**

parameters in the config file for the WCF service. Set **maxConcurrentCalls**, **maxConcurrentSessions**, and **maxConcurrentInstances** to a value greater than 16 * the number of CPUs or CPU cores. For example, on a computer with 8 CPU cores, set **maxConcurrentCalls**, **maxConcurrentSessions**, and **maxConcurrentInstances** to a value greater than 128 (16 * 8 = 128) as follows:

```
<serviceThrottling

maxConcurrentCalls="200"

maxConcurrentSessions="200"

maxConcurrentInstances="200" />
```

Increase the default values for the NetTcpBinding.ListenBacklog and NetTcpBinding.MaxConnections properties in the backend WCF service web.config file

The **NetTcpBinding.ListenBacklog** property controls the maximum number of queued connection requests that can be pending for a Web service. The **NetTcpBinding.MaxConnections** property controls the maximum number of connections to be pooled for subsequent reuse on the client and the maximum number of connections allowed to be pending dispatch on the server. Each of these properties uses a default value of 10 which may be suboptimal, especially for document processing scenarios that require high throughput.

Consider increasing the default value of these properties for high-throughput, document-processing scenarios that use WCF services which implement the **netTcpBinding** binding class.

In the following example, both the **listenBacklog** and **maxConnections** parameters are set to a value of “200”.

```
<netTcpBinding>

   <binding name="netTcpBinding"

      closeTimeout="00:10:00"

      openTimeout="00:10:00"

      receiveTimeout="00:10:00"

      sendTimeout="00:10:00"

      transactionFlow="false"

      transferMode="Buffered"

      transactionProtocol="OleTransactions"

      hostNameComparisonMode="StrongWildcard"

      listenBacklog="200"

      maxBufferPoolSize="1048576"

      maxBufferSize="10485760"

      maxConnections="200"
```

```
      maxReceivedMessageSize="10485760">
      <readerQuotas
         maxDepth="32"
         maxStringContentLength="8192"
         maxArrayLength="16384"
         maxBytesPerRead="4096"
         maxNameTableCharCount="16384" />
      <reliableSession
         ordered="true"
         inactivityTimeout="00:10:00"
         enabled="false" />
      <security mode="None">
         <transport clientCredentialType="Windows" protectionLevel="EncryptAndSign" />
         <message clientCredentialType="Windows" />
      </security>
   </binding>
</netTcpBinding>
```

For more information about the NetTcpBinding.ListenBacklog property, see NetTcpBinding.ListenBacklog Property (http://go.microsoft.com/fwlink/?LinkId=157752) on MSDN.

For more information about the NetTcpBinding.MaxConnections property, see NetTcpBinding.MaxConnections Property (http://go.microsoft.com/fwlink/?LinkID=157751) on MSDN.

Eliminate ASP.NET httpModules that are not required to run WCF Web services

By default, several ASP.NET httpModules are defined in the Request Pipeline in IIS 6.0 and in the Classic or Integrated Pipeline in IIS 7.5/7.0. These components intercept and process all incoming requests. The default modules are defined in the web.config file contained in the %windir%\Microsoft.NET\Framework\v2.0.50727\CONFIG folder for 32-bit ASP.NET applications and in the %windir%\Microsoft.NET\Framework64\v2.0.50727\CONFIG folder for 64-bit ASP.NET applications, as shown by the following snippet.

```
<httpModules>
<add name="OutputCache" type="System.Web.Caching.OutputCacheModule"/>
<add name="Session" type="System.Web.SessionState.SessionStateModule"/>
<add name="WindowsAuthentication"
type="System.Web.Security.WindowsAuthenticationModule"/>
```

```
<add name="FormsAuthentication" type="System.Web.Security.FormsAuthenticationModule"/>
<add name="PassportAuthentication"
type="System.Web.Security.PassportAuthenticationModule"/>
<add name="RoleManager" type="System.Web.Security.RoleManagerModule"/>
<add name="UrlAuthorization" type="System.Web.Security.UrlAuthorizationModule"/>
<add name="FileAuthorization" type="System.Web.Security.FileAuthorizationModule"/>
<add name="AnonymousIdentification"
type="System.Web.Security.AnonymousIdentificationModule"/>
<add name="Profile" type="System.Web.Profile.ProfileModule"/>
<add name="ErrorHandlerModule" type="System.Web.Mobile.ErrorHandlerModule,
System.Web.Mobile, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"/>
<add name="ServiceModel" type="System.ServiceModel.Activation.HttpModule,
System.ServiceModel, Version=3.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
</httpModules>
```

In most scenarios it is not necessary to load all of these modules. Therefore, it is possible to improve performance by eliminating the following httpModules when running WCF Web services:

- Session
- WindowsAuthentication
- FormsAuthentication
- PassportAuthentication
- RoleManager
- AnonymousIdentification
- Profile

Use the WCF Module/Handler Registration tool to configure WCF modules/handlers and improve scalability of IIS 7.5/7.0 hosted WCF services

The WCF Module/Handler Registration Tool is available for download at http://go.microsoft.com/fwlink/?LinkId=157593 (http://go.microsoft.com/fwlink/?LinkId=157593). This utility can be used to install, list, or configure the following WCF modules:

- WCF Synchronous HTTP module and handler
- WCF Asynchronous HTTP module and handler
- WCF HTTP module and handler

For more information about using the asynchronous WCF HTTP modules/handlers to improve the scalability of IIS 7.5/7.0 hosted WCF services, see Wenlong Dong's blog Orcas SP1 Improvement: Asynchronous WCF HTTP Module/Handler for IIS7 for Better Server Scalability (http://go.microsoft.com/fwlink/?LinkId=157428).

See Also

Optimizing BizTalk Server WCF Adapter Performance

Optimizing BizTalk Server Applications

This section describes methods for optimizing performance of BizTalk Server applications.

In This Section

- Optimizing Pipeline Performance
- Optimizing Memory Usage with Streaming
- Message Considerations

See Also

Optimizing Orchestration Performance

Optimizing Pipeline Performance

This topic describes guidelines for optimizing performance of pipelines in a BizTalk Server solution.

Best practices for optimizing performance of BizTalk Server pipelines

1. Because pipeline components have a significant impact on performance (for example, a pass-through pipeline component performs up to 30 percent better than an XML assembler/disassembler pipeline component), make sure that any custom pipeline components perform optimally before implementing them in your deployment. Minimize the number of pipeline components in your custom pipelines if you want to maximize the overall performance of your BizTalk application.
2. You also can improve overall performance by reducing the message persistence frequency in your pipeline component and by coding your component to minimize redundancy. Every custom assembly and in particular artifacts that could potentially disrupt performance, like custom tracking components, should be tested separately under heavy load condition to observe their behavior when the system is working at full capacity and to find any possible bottlenecks.
3. If you need to read the inbound message inside a pipeline component, avoid loading the entire document into memory using an **XmlDocument** object. The amount of space required by an instance of the **XmlDocument** class to load and create an in-memory representation of a XML document is up to 10 times the actual message size. In order to read a message, you should use an **XmlTextReader** object along with an instance of the following classes:
    - **VirtualStream (Microsoft.BizTalk.Streaming.dll)** - The source code for this class is located in two locations under the Pipelines SDK as follows: SDK\Samples\Pipelines\ArbitraryXPathPropertyHandler and SDK\Samples\Pipelines\SchemaResolverComponent\SchemaResolverFlatFileDasm.
    - **ReadOnlySeekableStream (Microsoft.BizTalk.Streaming.dll)**.

- **SeekAbleReadOnlyStream** - The source code for this class is located in two locations under the Pipelines SDK as follows: SDK\Samples\Pipelines\ArbitraryXPathPropertyHandler and SDK\Samples\Pipelines\SchemaResolverComponent\SchemaResolverFlatFileDasm.

4. Use the PassThruReceive and the PassThruTransmit standard pipelines whenever possible. They do not contain any pipeline component and do not perform any processing of the message. For this reason, they ensure maximum performance in receiving or sending messages. You can use a PassThruReceive pipeline on a receive location if you need to publish a binary document to the BizTalk MessageBox and a PassThruTransmit pipeline on a send port if you need to send out a binary message. You can also use the PassThruTransmit pipeline on a physical send port bound to an orchestration if the message has been formatted and is ready to be transmitted. You will need to use a different approach if you need to accomplish one of the following actions:
   - Promote properties on the context of an inbound XML or Flat File message.
   - Apply a map inside a receive location.
   - Apply a map in an orchestration that subscribes to a message.
   - Apply a map on a send port that subscribes to a message.

   To accomplish one of these actions, you must probe and discover the document type inside the receive pipeline and assign the (namespace#root-name) value to the MessageType context property. This operation is typically accomplished by a disassembler component such as the Xml Disassembler component (XmlDasmComp) or the Flat File disassembler component (FFDasmComp). In this case, you need to use a standard (for instance, XmlReceive pipeline) or a custom pipeline that contains a standard or a custom disassembler component.
5. Acquire resources as late as possible and release them as early as possible. For example, if you need to access data on a database, open the connection as late as possible and close it as soon as possible. Use the C# using statement to implicitly release disposable objects or the finally block of a try-catch-finally statement to explicitly dispose your objects. Instrument your source code to make your components simple to debug.
6. Eliminate any components from your pipelines that are not strictly required to speed up message processing.
7. Inside a receive pipeline, you should promote items to the message context only if you need them for message routing (Orchestrations, Send Ports) or demotion of message context properties (Send Ports).
8. If you need to include metadata with a message, and you don't use the metadata for routing or demotion purposes, use the **IBaseMessageContext.Write** method instead of the **IBaseMessageContext.Promote** method.
9. If you need to extract information from a message using an XPath expression, avoid loading the entire document into memory using an **XmlDocument** object just to use the **SelectNodes** or **SelectSingleNode** methods. Alternatively, use the techniques described in Optimizing Memory Usage with Streaming.

Use streaming to minimize the memory footprint required when loading messages in pipelines

The following techniques describe how to minimize the memory footprint of a message when loading the message into a pipeline.

Use ReadOnlySeekableStream and VirtualStream to process a message from a pipeline component

It is considered a best practice to avoid loading the entire message into memory inside pipeline components. A preferable approach is to wrap the inbound stream with a custom stream implementation, and then as read requests are made, the custom stream implementation reads the underlying, wrapped stream and processes the data as it is read (in a pure streaming manner). This can be very hard to implement and may not be possible, depending on what needs to be done with the stream. In this case, use the **ReadOnlySeekableStream** and **VirtualStream** classes exposed by the Microsoft.BizTalk.Streaming.dll. An implementation of these is also provided in Arbitrary XPath Property Handler (BizTalk Server Sample) (http://go.microsoft.com/fwlink/?LinkId=160069) in the BizTalk SDK.**ReadOnlySeekableStream** ensures that the cursor can be repositioned to the beginning of the stream. The **VirtualStream** will use a MemoryStream internally, unless the size is over a specified threshold, in which case it will write the stream to the file system. Use of these two streams in combination (using **VirtualStream** as persistent storage for the **ReadOnlySeekableStream**) provides both "seekability" and "overflow to file system" capabilities. This accommodates the processing of large messages without loading the entire message into memory. The following code could be used in a pipeline component to implement this functionality.

```
int bufferSize = 0x280;

int thresholdSize = 0x100000;

Stream vStream = new VirtualStream(bufferSize, thresholdSize);

Stream seekStream = new ReadOnlySeekableStream(inboundStream, vStream, bufferSize);
```

This code provides an "overflow threshold" by exposing bufferSize and thresholdSize variables on each receive location or send port configuration. This overflow threshold can then be adjusted by developers or administrators for different message types and different configurations (such as 32-bit vs. 64-bit).

Using XPathReader and XPathCollection to extract a given IBaseMessage object from within a custom pipeline component.

If specific values need to be pulled from an XML document, instead of using the **SelectNodes** and **SelectSingleNode** methods exposed by the XmlDocument class, use an instance of the XPathReader class provided by the Microsoft.BizTalk.XPathReader.dll assembly as illustrated in the following code example.

Note

For smaller messages especially, using an XmlDocument with SelectNodes or SelectSingleNode may provide better performance than using XPathReader, but XPathReader allows you to keep a flat memory profile for your application.

This example shows how to use the XPathReader and XPathCollection to extract a given **IBaseMessage** object from within a custom pipeline component.

```
public IBaseMessage Execute(IPipelineContext context, IBaseMessage message)
{
    try
    {
        ...
        IBaseMessageContext messageContext = message.Context;
        if (string.IsNullOrEmpty(xPath) && string.IsNullOrEmpty(propertyValue))
        {
            throw new ArgumentException(...);
        }
        IBaseMessagePart bodyPart = message.BodyPart;
        Stream inboundStream = bodyPart.GetOriginalDataStream();
        VirtualStream virtualStream = new VirtualStream(bufferSize, thresholdSize);
        ReadOnlySeekableStream readOnlySeekableStream = new
ReadOnlySeekableStream(inboundStream, virtualStream, bufferSize);
        XmlTextReader xmlTextReader = new XmlTextReader(readOnlySeekableStream);
        XPathCollection xPathCollection = new XPathCollection();
        XPathReader xPathReader = new XPathReader(xmlTextReader, xPathCollection);
        xPathCollection.Add(xPath);
        bool ok = false;
        while (xPathReader.ReadUntilMatch())
        {
            if (xPathReader.Match(0) && !ok)
            {
                propertyValue = xPathReader.ReadString();
                messageContext.Promote(propertyName, propertyNamespace, propertyValue);
                ok = true;
            }
        }
        readOnlySeekableStream.Position = 0;
        bodyPart.Data = readOnlySeekableStream;
    }
```

```
    catch (Exception ex)
    {
        if (message != null)
        {
            message.SetErrorInfo(ex);
        }
        ...
        throw ex;
    }
    return message;
}
```

Use XMLReader and XMLWriter with XMLTranslatorStream to process a message from a pipeline component

Another method for implementing a custom pipeline component which uses a streaming approach is to use the .NET **XmlReader** and **XmlWriter** classes in conjunction with the **XmlTranslatorStream** class provided by BizTalk Server. For example, the **NamespaceTranslatorStream** class contained in the Microsoft.BizTalk.Pipeline.Components assembly inherits from **XmlTranslatorStream** and can be used to replace an old namespace with a new one in the XML document contained in the stream. In order to use this functionality inside a custom a pipeline component, you can wrap the original data stream of the message body part with a new instance of the **NamespaceTranslatorStream** class and return the latter. In this way the inbound message is not read or processed inside the pipeline component, but only when the stream is read by a subsequent component in the same pipeline or is finally consumed by the Message Agent before publishing the document to the BizTalk Server MessageBox.

The following example illustrates how to use this functionality.

```
public IBaseMessage Execute(IPipelineContext context, IBaseMessage message)
{
   IBaseMessage outboundMessage = message;
   try
   {
      if (context == null)
      {
         throw new ArgumentException(Resources.ContextIsNullMessage);
      }
      if (message == null)
      {
```

```
            throw new ArgumentException(Resources.InboundMessageIsNullMessage);
        }

        IBaseMessagePart bodyPart = message.BodyPart;
        Stream stream = new NamespaceTranslatorStream(context,
                                                      bodyPart.GetOriginalDataStream(),
                                                      oldNamespace,
                                                      newNamespace);
        context.ResourceTracker.AddResource(stream);
        bodyPart.Data = stream;
    }
    catch (Exception ex)
    {
        if (message != null)
        {
            message.SetErrorInfo(ex);
        }

        throw ex;
    }
    return outboundMessage;
}
```

Using ResourceTracker in Custom Pipeline Components

A pipeline component should manage the lifetime of the objects it creates and perform garbage collection as soon as these objects are no longer required. If the pipeline component wants the lifetime of the objects to last until the end of pipeline execution, then you must add such objects to the resource tracker that your pipeline may fetch from the pipeline context.

The resource tracker is used for the following types of objects:

- Stream objects
- COM objects
- IDisposable objects

The message engine ensures that all the native resources that are added to the resource tracker are released at an appropriate time, that is after the pipeline is completely executed, regardless of whether it was successful or failed. The lifetime of the Resource Tracker instance and the objects that it is tracking is managed by the pipeline context object. The pipeline context is made

available to all types of pipeline components through an object that implements the IPipelineContext interface.

For example, the following code snippet is a sample that illustrates how to use ResourceTracker property in custom pipeline components. To use ResourceTracker property, the code snippet uses the following parameter `IPipelineContext.ResourceTracker.AddResource`. In this parameter:

- IPipelineContext interface defines the methods used to access all document processing-specific interfaces.
- ResourceTracker property references IPipelineContext and is used to keep track of objects that will be explicitly disposed at the end of pipeline processing.
- ResourceTracker.AddResource method is used to keep track of COM objects, Disposable objects and Streams, and should always be used inside a custom pipeline component to explicitly close (streams), dispose (IDisposable objects) or release (COM objects) these types of resources when a message is published to the BizTalk MessageBox.

```
public IBaseMessage Execute(IPipelineContext pContext, IBaseMessage pInMsg)


{
      IBaseMessage outMessage = pContext.GetMessageFactory().CreateMessage();

      IBaseMessagePart outMsgBodyPart = pContext.GetMessageFactory().CreateMessagePart();

      outMsgBodyPart.Charset = Encoding.UTF8.WebName;

      outMsgBodyPart.ContentType = "text/xml";

//Code to load message content in the MemoryStream object//

      MemoryStream messageData = new MemoryStream();

   //The MemoryStream needs to be closed after the whole pipeline has executed, thus
adding it into ResourceTracker//

      pContext.ResourceTracker.AddResource(messageData);

//Custom pipeline code to load message data into xmlPayload//

      XmlDocument xmlPayLoad = new XmlDocument();
```

```
        xmlPayLoad.Save(messageData);

        messageData.Seek(0, SeekOrigin.Begin);

//The new stream is assigned to the message part's data//

        outMsgBodyPart.Data = messageData;

// Pipeline component logic here//

        return outMessage;

}
```

Comparison of loading messages into pipelines using an in-memory approach and using a streaming approach

The following information was taken from Nic Barden's blog, http://blogs.objectsharp.com/cs/blogs/nbarden/archive/2008/04/14/developing-streaming-pipeline-components-part-1.aspx (http://go.microsoft.com/fwlink/?LinkId=160228). This table provides a summarized comparison of loading messages into pipelines using an in-memory approach and using a streaming approach.

Comparison of...	Streaming	In memory
Memory usage per message	Low, regardless of message size	High (varies depending on message size)
Common classes used to process XML data	Built in and custom derivations of: XmlTranslatorStream, XmlReader and XmlWriter	XmlDocument, XPathDocument, MemoryStream and VirtualStream
Documentation	Poor – many un-supported and un-documented BizTalk classes	Very good - .NET Framework classes
Location of "Processing Logic" code	• "Wire up" readers and streams via Execute method. • Actual execution occurs in the readers and streams as	Directly from the Execute method of the pipeline component.

Comparison of...	Streaming	In memory
	the data is read.	
Data	Re-created at each wrapping layer as data is read through it.	Read in, modified and written out at each component prior to next component being called.

See Also

Optimizing BizTalk Server Applications

Optimizing Memory Usage with Streaming

This topic provides recommendations for using streaming patterns to minimize message memory footprints when sending or receiving large messages with a WCF transport or when loading messages in orchestrations.

When using code in an orchestration to read the contents of a message, avoid using XmlDocument variables. Loading a message into an XmlDocument variable incurs significant overhead, especially for large messages. This overhead is in terms of memory usage and processing to build the in-memory structures. The use of an XmlDocument instance forces the entire message contents to be loaded into memory in order to build the object graph for the Document Object Module (DOM). The total amount of memory used by an instance of this class can be around 10 times the actual message size. For more information about the memory footprint required when loading a message into an XmlDocument variable, see Chapter 9 – Improving XML Performance (http://go.microsoft.com/fwlink/?LinkId=139772) on MSDN.

The remainder of this topic provides alternative methods for reading message contents that do not require loading a message into an XmlDocument variable.

Use streaming when sending or receiving large messages with a WCF transport

When sending or receiving large messages with a WCF transport, use the WCF-Custom or WCF-CustomIsolated adapter and configure with a binding type that supports the **transferMode = Streamed** option, such as the following bindings:

- **basicHttpBinding + BasicHttpBindingElement, transferMode = Streamed**
- **netTcpBinding + NetTcpBindingElement, transferMode = Streamed**
- **customBinding + HttpTransportElement, transferMode = Streamed**
- **customBinding +ConnectionOrientedTransportElement, transferMode = Streamed**

Choosing a WCF-Custom or WCF-CustomIsolated adapter along with a binding that supports the **transferMode = Streamed** option will implement streaming of large messages to the file system as needed, and will mitigate potential out-of-memory issues.

Use streaming to minimize the memory footprint required when loading messages in orchestrations

The following techniques describe how to minimize the memory footprint of a message when loading the message into an orchestration.

Use an XLANGMessage variable to process the contents of a message or message part

When passing a message from an orchestration to .NET class libraries, do not pass them as XmlDocument variables, for reasons mentioned earlier in this topic; use XLANGMessage variables instead. The following techniques illustrate methods for reading a message or message part using an XLANGMessage variable.

- **Process messages with XMLReader** - To process a message with an XmlReader instance, pass the message to .NET code as an XLANGMessage, and retrieve the Part content using XmlReader.

```
public void ProcessMessage(XLANGMessage message)
{
    try
    {
        using (XmlReader reader =
message[0].RetrieveAs(typeof(XmlReader)) as XmlReader)
        if (reader != null)
        {
            ...
        }
    }
    finally
    {
        message.Dispose();
    }
}
```

- **Retrieve the contents of a message into a string with StreamReader** - One of the common uses of XmlDocument in orchestrations is to retrieve the message as an XML string using XmlDocument.OuterXml(). The following code example illustrates an alternative method which retrieves the message as a string using an XLANGMessage variable.

```
public static string MessageToString(XLANGMessage message)
{
    string strResults;
    try
    {
        using (Stream stream =
message[0].RetrieveAs(typeof(Stream)) as Stream)
        {
```

```
            using (StreamReader reader = new
StreamReader(stream))
            {
                strResults = reader.ReadToEnd();
            }
        }
    }
    finally
    {
        message.Dispose();
    }
    return strResults;
}
```

- **Retrieve the contents of simple .NET messages into a string** - If the type of the message is a simple .NET type, you can retrieve the message as that type. For example, to get the message as a string, pass the message to the .NET code as an XLANGMessage and retrieve the Part content as a string.

```
public void ProcessMessage(XLANGMessage message)
{
    try
    {
        string content = message[0].RetrieveAs(typeof(string))
as string;
        if (!string.IsNullOrEmpty(content))
        {
            ...
        }
    }
    finally
    {
        message.Dispose();
    }
}
```

- **Retrieve the contents of a message into a stream** - To get the message as a stream, pass the message to the .NET code as an XLANGMessage and retrieve the Part content as a stream.

```
public Stream ProcessRequestReturnStream(XLANGMessage message,
int bufferSize, int thresholdSize)
{
   ...
   try
   {
      using (VirtualStream virtualStream = new
VirtualStream(bufferSize, thresholdSize))
      {
         using (Stream partStream =
(Stream)message[0].RetrieveAs(typeof(Stream)))
         //Note that when calling this code, if the XmlDocument
is quite large, keeping it in a memory with a MemoryStream may
have an adverse effect on performance.
         //In this case, it may be worthwhile to consider an
approach that uses a VirtualStream + ReadonlySeekableStream to
buffer it to the file system, if its size is bigger than the
thresholdSize parameter.
         //Keep in mind that:
         // - If the message size is smaller than the threshold
size, the VirtualStream class buffers the stream to a
MemoryStream.
         // - If the message size is bigger than the threshold
size, the VirtualStream class buffers the stream to a temporary
file.
            using (ReadOnlySeekableStream readOnlySeekableStream
= new ReadOnlySeekableStream(partStream, virtualStream,
bufferSize))
            {
               using (XmlReader reader =
XmlReader.Create(readOnlySeekableStream))
               {

               }
            }
         }
```

```
        }
    }
    catch (Exception ex)
    {

    }
    finally
    {
        message.Dispose();
    }
    return stream;
}
```

- **Retrieve the contents of a message into a .NET object** - To get the message as a .NET object, pass the message to the .NET code as an XLANGMessage and retrieve the Part content as an instance of a .NET class. Create this latter from the Xml Schema of the message using the XML Schema Definition Tool (Xsd.exe) tool provided by Visual Studio 2010.

  Note
  This technique is valid only when messages are small. Otherwise, this approach could incur in a significant overhead to de-serialize the actual message into a .NET object.

```
public void ProcessMessage(XLANGMessage message)
{
    try
      {
      Request request = message[0].RetrieveAs(typeof(Request))
as Request;
      if (request != null)
      {
         ...
      }
    }
    finally
    {
      message.Dispose();
```

```
        }

    }
```

Note

Use of the **Dispose()** method exposed by the XLANGMessage parameter before returning from the .NET code is particularly important in looping scenarios and long-running orchestrations that can accumulate instances of the XLANGMessage object without releasing them over time. For more information about calling message.Dispose() in this manner, see Messages Represented as XLANGMessage (http://go.microsoft.com/fwlink/?LinkId=139775) in the BizTalk Server documentation. This topic also provides best practices for using IStreamFactory to construct XLANGMessage variables in user code using a stream-based approach.

For more information about the different ways to process an XLANGMessage within an helper component invoked by an orchestration, see the following topics:

- 4 Different ways to process an XLANGMessage within an helper component invoked by an orchestration Part 1 (http://go.microsoft.com/fwlink/?LinkID=210420).
- 4 Different ways to process an XLANGMessage within an helper component invoked by an orchestration Part 2 (http://go.microsoft.com/fwlink/?LinkID=210421).

Using XPathReader and XPathCollection to extract a value from an XLANGMessage object from a method invoked by an orchestration

Avoid using the XMLDocument class to read the contents of XML messages from custom code, such as custom pipeline components or helper classes invoked by orchestrations. When using an XMLDocument instance to load an XML message, the entire message is loaded into memory, which is inefficient and may require memory up to 10 times the actual size of the message. A more efficient way of reading the contents of XML messages is to use a streaming technique to wrap the original stream with one of the stream classes provided by the Microsoft.BizTalk.Streaming.dll assembly. This technique is particularly useful when loading large messages.

If specific values need to be pulled from an XML document, instead of using the SelectNodes and SelectSingleNode methods exposed by the XmlDocument class, use an instance of the XPathReader class provided by the Microsoft.BizTalk.XPathReader.dll assembly as illustrated in the following code example.

- This example illustrates using the XPathReader and XPathCollection to extract a given value from an XLANGMessage object inside a method invoked by an orchestration.

```
public static string SelectSingleNode(XLANGMessage message,
string xPath)
{
    try
    {
```

```
            if (message == null || string.IsNullOrEmpty(xPath))
            {
                return string.Empty;
            }
            using (XmlReader reader =
    (XmlReader)message[0].RetrieveAs(typeof(XmlReader)))
            {
                XPathCollection xPathCollection = new
    XPathCollection();
                XPathReader xPathReader = new XPathReader(reader,
    xPathCollection);
                xPathCollection.Add(xPath);
                while (xPathReader.ReadUntilMatch())
                {
                    if (xPathReader.Match(0))
                    {
                        return xPathReader.ReadString();
                    }
                }
            }
        }
        catch (Exception ex)
        {
            ...
        }
        finally
        {
            message.Dispose();
        }
        return string.Empty;
    }
```

See Also

Optimizing BizTalk Server Applications

Message Considerations

BizTalk Server provides an extensive set of capabilities for sending, receiving, transforming, and processing messages. Some of these capabilities include:

- Ability to send and receive messages using multiple industry standard transports such as HTTP, SMTP, FTP/FTPS and WCF. Transport level support for sending and receiving messages is accomplished using BizTalk Server adapters. Integration of BizTalk Server message processing with various “line of business” (LOB) applications is accomplished using one of many available BizTalk Server accelerators or adapters, such as the BizTalk Accelerator for HIPAA, BizTalk Accelerator for SWIFT, or the BizTalk SAP adapter. These accelerators and adapters conform to industry standards which in turn accommodates straightforward integration of BizTalk Server with systems that conform to particular industry standard.
- Ability to process multiple message formats, such as plain text, XML, binary, and others. This functionality is crucial to providing integration of BizTalk Server with a broad spectrum of trading partners.
- Support for message transformation or document mapping. Mapping accommodates the transformation of data between different schemas. For example, mapping could be used to transform the contents of a received customer purchase order into a receipt with shipping notification to be sent back to the customer.
- BizTalk Server orchestrations provide functionality for creating business processes that span time, organizations, applications, and people. BizTalk Server provides the Orchestration Designer graphical interface to develop business processes that include support for transactions (both traditional “atomic” MSDTC type and long running), exception handling, debugging, tracking, and extensibility for calling external code.

  Note
  See Optimizing Orchestration Performance for guidance about best practices to follow when using orchestrations in BizTalk Server. See the topic Creating Orchestrations Using Orchestration Designer (http://go.microsoft.com/fwlink/?LinkId=158997) in the BizTalk Server documentation for in-depth information about using Orchestration Designer.

The remainder of this topic describes performance considerations related to the size, complexity, and distribution profile of messages that are processed in a BizTalk Server 2010 environment.

Message size considerations

While BizTalk Server imposes no restriction on message size, practical limits and dependencies might require you to minimize the size of your messages because large messages require more processing resources. As message size increases, overall throughput (messages processed per second) decreases. When designing your scenario and planning for capacity, consider the average message size, message type, and number of messages BizTalk Server processes. Do not use unnecessarily long attribute and tag names; if possible, keep the length under 50 characters. For example, do not use a 200-character tag name for a message size of only 1 byte.

If the in-memory size of a received message exceeds the number of bytes specified for the Large message fragment size (configurable on the Group Properties page for the BizTalk Group in the

BizTalk Server Administration console), the message is split into fragments of the specified size. Additionally, the fragments are written into the Messagebox under the context of a Microsoft Distributed Transaction Coordinator (MSDTC) transaction as follows:

1. If the incoming message is being published in the context of an existing MSDTC transaction, this transaction is used when writing the message fragments to the BizTalk MessageBox database. For example, if the incoming message is being published by a transactional adapter configured to require transactions, the existing transaction would be used when writing the message fragments to the MessageBox database.
2. If the incoming message is not being published in the context of an existing MSDTC transaction, a new MSDTC transaction is created to write the message fragments to the MessageBox database. In this scenario, the following considerations apply:
    - Increase the value for **Large message fragment size** to reduce the frequency with which large messages are fragmented and reduce the incidence of creating the associated MSDTC transactions. This should be done because excessive use of MSDTC transactions is expensive from a performance standpoint. Note that increasing this value may also increase the amount of available memory that is used.
    - If it takes longer than the maximum allowable MSDTC transaction timeout of 60 minutes to write a message to the MessageBox, the transaction times out, an error occurs, and the attempt to write the message fails and is rolled back. The **Large message fragment size** value should be increased enough to avoid this problem when processing very large messages. Depending on available memory, this value should be increased up to a maximum value of 1000000 bytes.
    - Each message fragment in a message creates one or more SQL Server database locks against the MessageBox database. When the number of locks exceeds several hundred thousand, it is possible that SQL Server will generate “out of lock” errors. If this problem occurs, increase the value for **Large message fragment size** to reduce the number of fragments (which decreases the number of SQL Server database locks made against the MessageBox database) or consider housing your MessageBox database on a 64-bit version of SQL Server. The number of available locks is significantly higher on the 64-bit version of SQL Server than on a 32-bit version of SQL Server. The following formula can be used to estimate the maximum number of messages per interchange when the MessageBox database is housed on a 32-bit version of SQL Server:

      ```
      200,000 / (Number of CPUs * BatchSize * MessagingThreadPoolSize)
      ```

For more information about how BizTalk Server processes large messages, including guidelines for processing large messages, see [How BizTalk Server Processes Large Messages](http://go.microsoft.com/fwlink/?LinkID=154680) (http://go.microsoft.com/fwlink/?LinkID=154680).

Message type considerations

Messages are received into BizTalk Server in one of two primary formats: XML files or flat files. Message type should always be factored into the message distribution profile because of the varying resource requirements of XML and flat file messages.

- **XML files** In order for BizTalk Server to perform any processing on a message other than pass-through routing, the message must be in the XML file format.

- **Flat files** Flat files must be parsed into an XML format before BizTalk Server can perform any processing other than pass-through routing. Parsing a flat file into an XML file can greatly increase the size of the file. Flat files do not contain XML tags with descriptive information about their data. XML files, on the other hand, wrap all of their data in descriptive XML tags. In some scenarios, parsing can increase the size of a flat file by a factor of 10 or more, depending on how much descriptive data is contained in the XML tags for the file.
- **Flat file documents wrapped in a single CDATA section node in an XML document** This type of document is a combination of both XML and flat file, and is often memory-intensive, since BizTalk Server must load the entire wrapped flat file document into memory before processing it.

Overload considerations

Most BizTalk Server applications do not receive messages at a specific, constant rate. Typically, message processing is subject to peaks and valleys. , For example, an online banking application may process a higher volume of messages first thing in the morning, then during the middle of the day, and at the end of the day. BizTalk Server solutions should be tested to ensure that they will be able to handle these overload scenarios. To determine how well a BizTalk Server solution can handle overload scenarios, the maximum sustainable throughput (MST) of the BizTalk Server solution should be determined. The MST is the highest load of message traffic that a system can handle indefinitely in a production environment. For more information about MST, see Planning for Sustained Performance (http://go.microsoft.com/fwlink/?LinkID=158065) and What Is Sustainable Performance? (http://go.microsoft.com/fwlink/?LinkID=132304) in the BizTalk Server documentation.

Schema complexity

The throughput for message parsing (especially flat-file parsing) is affected by the complexity of the schemas. As schema complexity increases, overall performance decreases. When designing schemas, reduce the length of node names, and move promoted properties to the top of the schema. This reduces retrieval time and thereby increases performance.

Map complexity

Depending on the complexity of the maps, map transformation can be resource-intensive. As map complexity increases, overall performance decreases. To improve overall performance, minimize the number of links and functoids used in your maps, especially functoids that call external resources such as the DB Lookup functoid.

Flat-file parsing considerations

The two factors that have the highest impact on the performance of flat-file parsing are file size and schema complexity. An ambiguous schema is a schema that contains many optional fields. When large file sizes are used, a schema with many optional fields can degrade performance because larger files may match different branches of the schema. Schema complexity has less impact on smaller files than on larger files.

See Also

Optimizing BizTalk Server Applications

Windows PowerShell Scripts

This topic contains Windows PowerShell scripts that can be run on the computers in a BizTalk Server environment to apply registry settings described in this guide.

Important

These scripts should only be run on Windows Server 2003, not on Windows Server 2008 SP2. While these scripts will execute successfully on Windows Server 2008 SP2, lab testing has indicated that these scripts do not provide any significant performance advantage on Windows Server 2008 SP2. These scripts should only be run on a Windows Server 2003 production environment after thorough testing and evaluation.

Optimizing operating system performance through registry settings

The following Windows PowerShell script can be used to apply the registry settings described in Optimizing Operating System Performance.

Copy the script below into Notepad and save as Set-OSRegSettings.ps1. Then run the script on each computer in the BizTalk Server environment by following the instructions in Optimizing Operating System Performance:

```
#Set-OSRegSettings.ps1

param($RAMMb, $NumCPU,$VolType)

if (($RAMMb -eq $null) -or ($NumCPU -eq $null) -or ($VolType -eq $null) -or ($VolType -gt
4))

{

"`r"

"Please specify required paramemters of -RAMMb and -NumCPU and -VolType"

"Usage: .\OSSettings.ps1 -RAMMb 2048 -NumCPU 2 -VolType 4"

"Valid VolType values are: 1(few files), 2 or 3(moderate files), 4(many files)"

"`r"

exit

}

$ErrorActionPreference = "SilentlyContinue"

$LogFileName="OSSettings.log"

$LogTime=([System.DateTime]::Now).ToString("dd-MM-yyyy hh:mm:ss")

Add-Content $LogFileName "*********** Settings changed at $LogTime ************"
```

```
function SetProperty([string]$path, [string]$key, [string]$Value)
{
#Clear Error Count
$error.clear()
$oldValue = (Get-ItemProperty -path $path).$key
#Set the Registry Key
Set-ItemProperty -path $path -name $key -Type DWORD -Value $Value
#if error count is 0, regkey was updated OK
if ($error.count -eq 0)
{
$newValue = (Get-ItemProperty -path $path).$key
$data =  "$path\$key=$oldValue"
if($oldvalue -eq $null)
{
Write-Output "Value for $path\$key created and set to $newValue"
Add-Content $LogFileName "Value for $path\$key created and set to $newValue"
}
else
{
Write-Output "Value for $path\$key changed from $oldValue to $newValue"
Add-Content $LogFileName "Value for $path\$key changed from $oldValue to $newValue"
}
}
#if error count is greater than 0 an error occurred and the regkey was not set
else
{
Add-Content $LogFileName "Error: Could not set key $path\$key"
Add-Content $LogFileName $Error[$error.count-1].exception.message
Write-Output "Error: Could not set key $path\$key"
Write-Output $Error[$error.count-1].exception
}
}

SetProperty "HKLM:\SYSTEM\CurrentControlSet\Services\ldap" "ldapclientintegrity" 0x0
```

```
#Work out Value of $IoPageLockLimit
$IoPageLockLimit = ($RAMMb - 65) * 1024
if ($IoPageLockLimit -gt 4294967295)
{
$IoPageLockLimit = "0xFFFFFFFF"
}
else
{
#Convert to Hexadecimal

$IoPageLockLimit = "{0:X}" -f $IoPageLockLimit
$IoPageLockLimit = "0x" + $IoPageLockLimit
}
SetProperty "HKLM:\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management"
"IoPageLockLimit" ($IoPageLockLimit)

#Work out Value of $WorkerThreads
$WorkerThreads = $NumCPU * 16
if ($WorkerThreads -gt 64)
{$WorkerThreads = "0x40"} #Hexadecimal Value of 64
else
{
$WorkerThreads = "{0:X}" -f $WorkerThreads
$WorkerThreads = "0x" + $WorkerThreads
}

SetProperty "HKLM:\SYSTEM\CurrentControlSet\Control\Session Manager\Executive"
"AdditionalDelayedWorkerThreads" 0x10
SetProperty "HKLM:\SYSTEM\CurrentControlSet\Control\Session Manager\Executive"
"AdditionalCriticalWorkerThreads" 0x10
SetProperty "HKLM:\SYSTEM\CurrentControlSet\Services\LanmanServer\Parameters"
"MaxWorkItems" 8192
SetProperty "HKLM:\SYSTEM\CurrentControlSet\Services\LanmanServer\Parameters" "MaxMpxCt"
2048
```

```
SetProperty "HKLM:\SYSTEM\CurrentControlSet\Services\LanmanWorkstation\Parameters"
"MaxCmds" 2048

#Value depends of -VolType parameter passed in

SetProperty "HKLM:\SYSTEM\CurrentControlSet\Control\FileSystem" "NtfsMftZoneReservation"
$VolType

SetProperty "HKLM:\SYSTEM\CurrentControlSet\Control\FileSystem"
"NTFSDisableLastAccessUpdate" 1

SetProperty "HKLM:\SYSTEM\CurrentControlSet\Control\FileSystem"
"NTFSDisable8dot3NameCreation" 1

SetProperty "HKLM:\SYSTEM\CurrentControlSet\Control\FileSystem" "DontVerifyRandomDrivers"
1

SetProperty "HKLM:\System\CurrentControlSet\Services\LanmanServer\Parameters" "Size" 3

SetProperty "HKLM:\System\CurrentControlSet\Control\Session Manager\Memory Management"
"LargeSystemCache" 0
```

Optimizing network performance through registry settings

The following Windows PowerShell script can be used to apply the registry settings described in Optimizing Network Performance.

Copy the script below into Notepad and save as Set-NetworkRegSettings.ps1. Then run the script on each computer in the BizTalk Server environment by following the instructions in Optimizing Network Performance:

```
#Set-NetworkRegSettings.ps1

param([int] $MTUSize = $(throw "usage: ./Set-NetworkSettings <MTU Size>"))



$LogFileName="NetworkRegSettings.log"

$LogTime=([System.DateTime]::Now).ToString("dd-MM-yyyy hh:mm:ss")

Add-Content $LogFileName "*********** Settings changed at $LogTime ************"


function SetProperty([string]$path, [string]$key, [string]$Value) {

$oldValue = (Get-ItemProperty -path $path).$key

Set-ItemProperty -path $path -name $key -Type DWORD -Value $Value

$newValue = (Get-ItemProperty -path $path).$key

$data =  "$path\$key=$oldValue"

Add-Content $LogFileName $data

Write-Output "Value for $path\$key changed from $oldValue to $newValue"
```

```
}
SetProperty "HKLM:\System\CurrentControlSet\Control\Session Manager\Memory Management"
"DisablePagingExecutive" 1
# Set SystemPages to 0xFFFFFFFF
# Maximize system pages. The system creates the largest number of page table entries
possible within physical memory.
# The system monitors and adjusts this value dynamically when the configuration changes.
SetProperty "HKLM:\System\CurrentControlSet\Control\Session Manager\Memory Management"
"SystemPages" 0xFFFFFFFF
# HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\LanmanServer\Parameters
# ============================================================================
$path = "HKLM:\System\CurrentControlSet\Services\LanmanServer\Parameters"
$returnValue = (Get-ItemProperty -path $path).IRPStackSize
if ( $returnValue -eq $null) {
SetProperty $path "IRPStackSize" 0x20 # IRPStackSize -> +10 (Use DWORD 0x20 (32) if not
present)
}else{
$returnValue = $returnValue + 1
SetProperty $path "IRPStackSize" $returnValue
}
SetProperty $path "SizReqBuf" 0x4000 # SizReqBuf -> 0x4000 (16384)

# HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
# ================================================================
$path = "HKLM:\System\CurrentControlSet\Services\Tcpip\Parameters"
SetProperty $path "DefaultTTL" 0x40 # DefaultTTL -> 0x40 (64)
SetProperty $path "EnablePMTUDiscovery" 1 # EnablePMTUDiscovery -> 1 (do not enable this
if your server is directly exposed to potential attackers)
SetProperty $path "EnablePMTUBHDetect" 1 # EnablePMTUBHDetect -> 1 (if your system is
using a SOAP or HTTP and/or initiating web connections to other systems)
SetProperty $path "TcpMaxDupAcks" 2 # TcpMaxDupAcks -> 2
SetProperty $path "Tcp1323Opts" 1 # Tcp1323Opts -> 1 (experiment with a value of 3 for
possible improved performance, especially if you are experiencing high packet
loss/retransmits)
```

```
SetProperty $path "SackOpts" 1 # SackOpts -> 1 (VERY important for large TCP window sizes, such as specified below)

SetProperty $path "MaxFreeTcbs" 0x5000 # MaxFreeTcbs -> 0x5000 (20480)

SetProperty $path "TcpMaxSendFree" 0xFFFF # TcpMaxSendFree -> 0xFFFF (65535)

SetProperty $path "MaxHashTableSize" 0xFFFF # MaxHashTableSize -> 0xFFFF (65535)

SetProperty $path "MaxUserPort" 0xFFFF # MaxUserPort -> 0xFFFF (65535)

SetProperty $path "TcpTimedWaitDelay" 0x1E # TcpTimedWaitDelay -> 0x1E (30)

SetProperty $path "GlobalMaxTcpWindowSize" 0xFFFF # GlobalMaxTcpWindowSize -> 0xFFFF (65535)

SetProperty $path "NumTCPTablePartitions" 4 # NumTCPTablePartitions -> 2 per Processor/Core (include processor cores BUT NOT hyperthreading)

# TcpAckFrequency (requires Windows Server 2K3 Hotfix 815230)

# SetProperty $path "TcpAckFrequency" 5 #5 for 100Mb, 13 for 1Gb (requires Windows Server 2K3 Hotfix 815230) - can also be set at the interface level if mixed speeds; only set for connections primarily processing data

SetProperty $path "SynAttackProtect" 0 # SynAttackProtect -> 0 (Only set this on systems with web exposure if other H/W or S/W is providing DOS attack protection)

#Dedicated Network (DATA)

#------------------------

#Interfaces\<adapter ID>\MTU -> 1450-1500, test for maximum value that will pass on each interface using PING -f -l <MTU Size> <Interface Gateway Address>, pick the value that works across all interfaces

$RegistryEntries = Get-ItemProperty -path "HKLM:\system\currentcontrolset\services\tcpip\parameters\interfaces\*"

foreach ( $iface in $RegistryEntries ) {

$ip = $iface.DhcpIpAddress

if ( $ip -ne $null ) { $childName = $iface.PSChildName}

else {

$ip = $iface.IPAddress

if ($ip -ne $null) { $childName = $iface.PSChildName }

}

$Interface = Get-ItemProperty -path "HKLM:\system\currentcontrolset\services\tcpip\parameters\interfaces\$childName"

$path = $Interface.PSPath

SetProperty $path MTU $MTUSize
```

```
}
$path = "HKLM:\System\CurrentControlSet\Services\Tcpip\Parameters"
$ForwardBufferMemory = 100*$MTUSize
SetProperty $path "ForwardBufferMemory" $ForwardBufferMemory # ForwardBufferMemory ->
100*MTUSize, rounded up to the nearest 256 byte boundary
SetProperty $path "MaxForwardBufferMemory" $ForwardBufferMemory # MaxForwardBufferMemory
-> ForwardBufferMemory
$NumForwardPackets = $ForwardBufferMemory/256
SetProperty $path "NumForwardPackets" $NumForwardPackets # NumForwardPackets ->
ForwardBufferMemory / 256
SetProperty $path "MaxNumForwardPackets" $NumForwardPackets # MaxNumForwardPackets ->
NumForwardPackets
SetProperty $path "TcpWindowSize" 0xFBA4 # TcpWindowSize -> 0xFBA4 (64420) (make this a
multiple of the TCP MSS (Max Segment Size)
# HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\AFD\Parameters
# ================================================================
$path = "HKLM:\SYSTEM\CurrentControlSet\Services\AFD\Parameters"
SetProperty $path "EnableDynamicBacklog" 1 # EnableDynamicBacklog -> 1
SetProperty $path "MinimumDynamicBacklog" 0xc8 # MinimumDynamicBacklog -> 0xc8 (200)
SetProperty $path "MaximumDynamicBacklog" 0x4e20 # MaximumDynamicBacklog -> 0x4e20
(20000)
SetProperty $path "DynamicBacklogGrowthDelta" 0x64 # DynamicBacklogGrowthDelta -> 0x64
(100)
#S/W Configuration
#=====================================================================
#Disable NETBIOS on cluster private network, if configured
```

See Also

Optimizing Performance

Scaling a Production BizTalk Server Environment

This section provides an overview of the lab environment that was used to perform load testing of a BizTalk Server solution, a summary of the load test results, and general observations and recommendations based upon the findings in the lab.

The information provided in these topics can and should be used as a guide for scaling your BizTalk Server environment. In addition to this guidance we recommend that you complete periodic load testing throughout the development cycle of your BizTalk Server application.

In This Section

- Scenario Overview
- Key Performance Indicators
- Observations and Recommendations

Scenario Overview

This topic provides an overview of load testing completed by the BizTalk Server product group to assess scalability of BizTalk Server 2010 when running on modern enterprise class hardware.

All testing was performed in an isolated environment using dedicated hardware. Over 200 test runs were performed and all results were validated by the BizTalk Server product group.

Tests were performed using a Messaging scenario and an Orchestration scenario; both scenarios utilized the BizTalk WCF-NetTcp adapter.

To evaluate the maximum possible performance of the BizTalk Server engine, no custom pipeline components were used and only a single, very simple orchestration was used for the Orchestration scenario. Performance optimizations described in Optimizing Performance were applied to the environment and are fully documented in Observations and Recommendations.

Test Goals

The goals of the load testing performed included the following:

1. Provide general sizing and scaling guidance for BizTalk Server 2010:
    a. Quantify the impact of adding additional computers running BizTalk Server to a BizTalk Server group. For this testing, the performance of a BizTalk Server solution was measured when the BizTalk Server group was running one, two, three, and four computers running BizTalk Server.
    b. Quantify the impact of adding additional BizTalk MessageBox databases to a BizTalk Server group. For this testing, the performance of a BizTalk Server solution was measured when the BizTalk Server group was configured to use either a single MessageBox database or four MessageBox databases.

> **Note**
> Testing with two MessageBox databases was not done because there is little, if any, performance advantage when scaling from one to two MessageBox databases. In fact, scaling from one to two MessageBox databases can adversely affect performance. For more information about scaling out the BizTalk Server MessageBox, see Scaling Out the SQL Server Tier (http://go.microsoft.com/fwlink/?LinkID=158075) in the BizTalk Server documentation.

2. Provide sizing and scaling guidance for the following scenarios:
    a. WCF-NetTcp one-way Messaging scenario
    b. WCF-NetTcp one-way Orchestration scenario

Test Measurements Used

BizTalk Server performance was measured using the following criteria:

1. **Overall throughput** –Measured with the **BizTalk:Messaging(hostname)\Documents received/Sec** and **BizTalk:Messaging(hostname)\Documents processed/Sec** performance counters for the BizTalk Server receive and processing hosts.
2. **CPU utilization** – measured with the **\Processor(_Total)\%Processor Time** performance counters on the BizTalk Server and SQL Server computers. All test results were comprehensively reviewed and any performance bottlenecks are described in Observations and Recommendations.

Scaling out the processing tier and the database tier

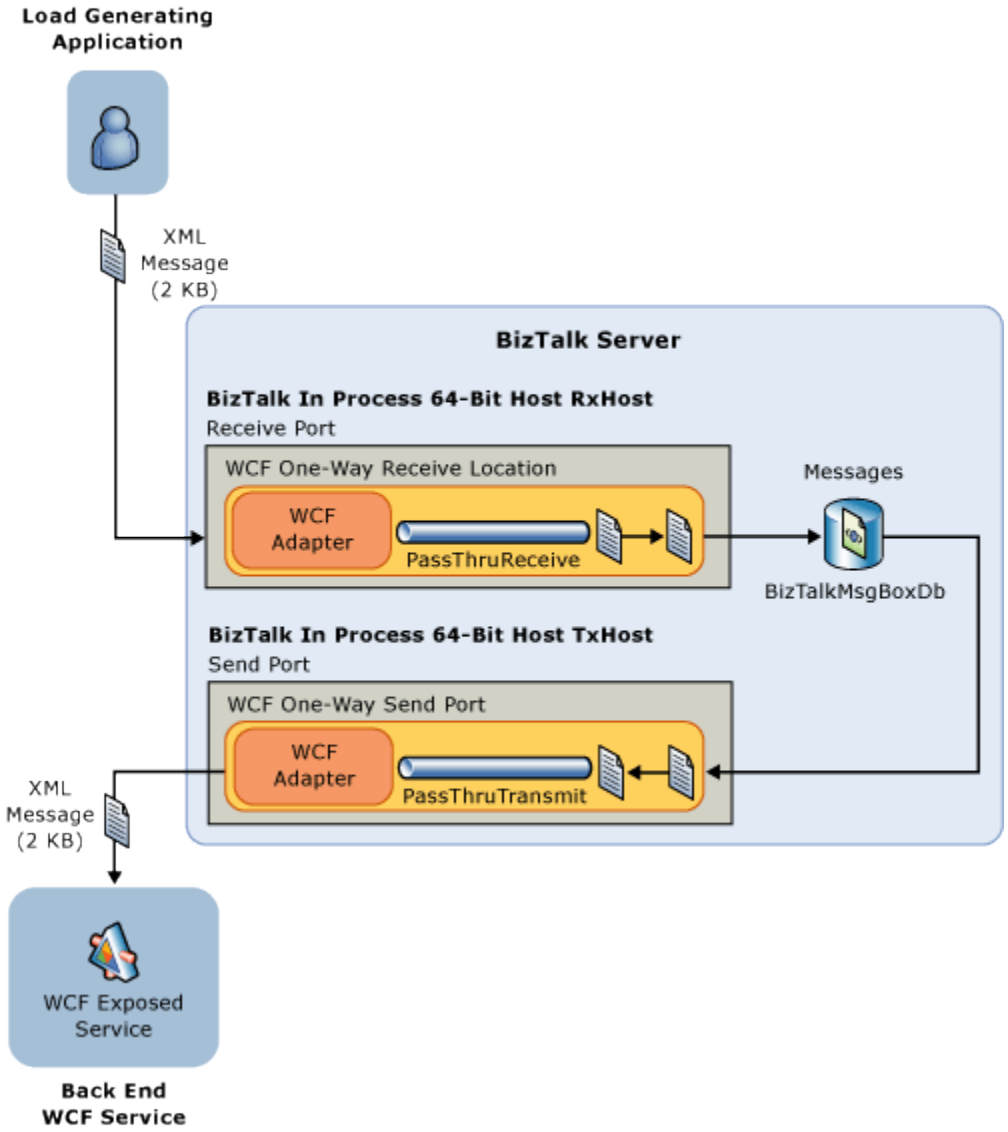
BizTalk Server easily accommodates increased processing tier capabilities by adding one or more BizTalk Server computers to an existing BizTalk Server group. BizTalk Server accommodates increased database tier capabilities through the addition of MessageBox databases.

To provide scale out metrics for BizTalk Server, tests were performed with one, two, three, and four BizTalk Server computers. To demonstrate the impact of scaling out the database tier, these tests were performed against both single and multi-MessageBox systems.

Testing scenarios

The flow of messages through the BizTalk Server environment for these scenarios is described in detail below.

Messaging test scenario



1. The Visual Studio 2010 Ultimate edition load testing functionality generates an XML message and sends it to the computer running BizTalk Server with the NetTcp transport.

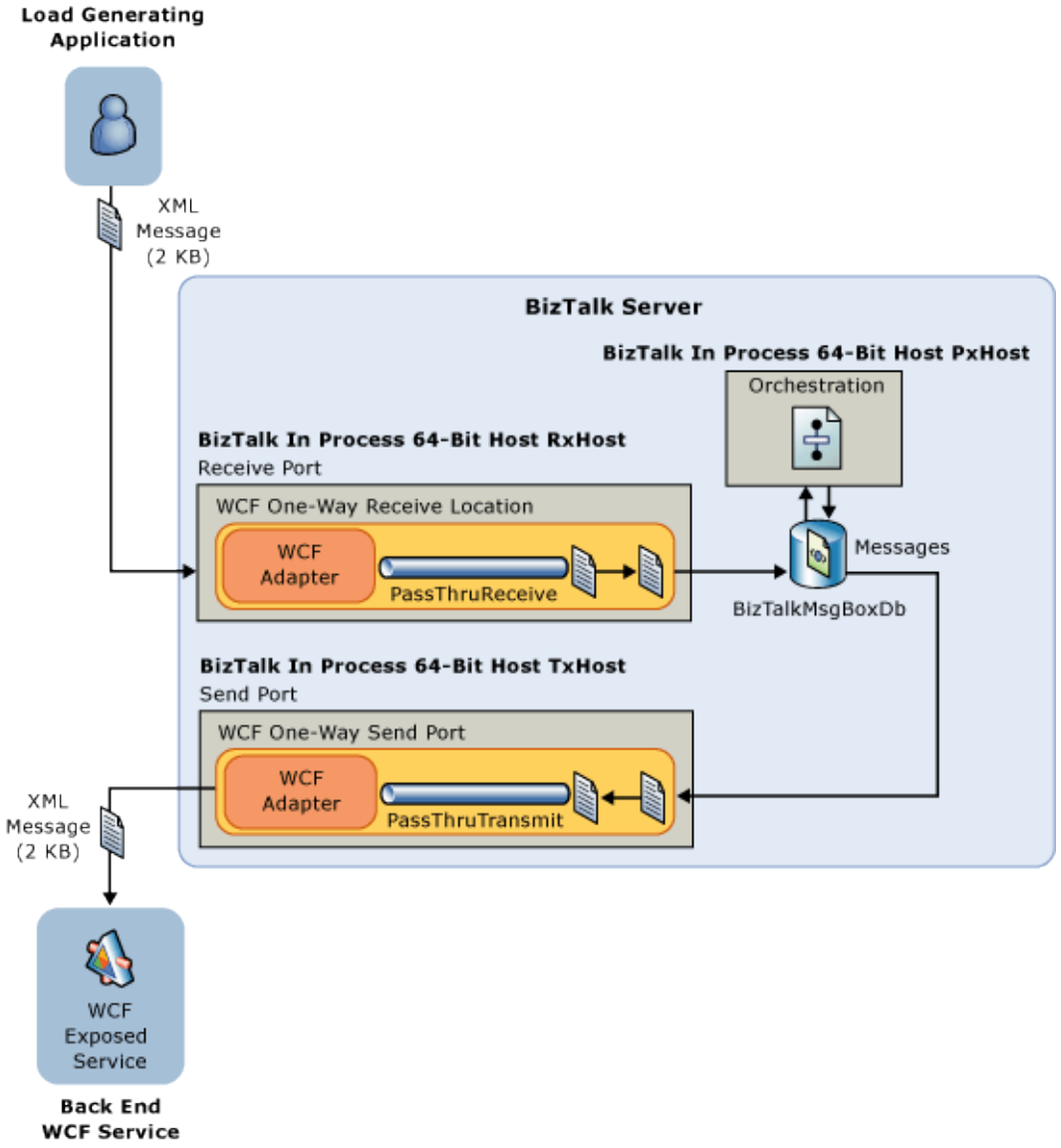
   Note

   For more information about Visual Studio 2010 Ultimate edition load testing, see Testing the Application (http://go.microsoft.com/fwlink/?LinkID=208247).

   For more information about how we used the Visual Studio 2010 Ultimate edition load testing functionality in our test environment, see Using Visual Studio to Facilitate Automated Testing.

2. The XML message is received by a BizTalk Server receive location that uses the WCF-NetTcp receive adapter. The BizTalk Server receive location is configured to use the PassThruReceive pipeline, which performs no processing of the message.
3. The BizTalk Server end point manager (EPM) publishes the message to the BizTalk MessageBox database.
4. A BizTalk Server send port that uses the WCF-NetTcp send adapter subscribes to messages published by the receive location and retrieves the message from the BizTalk MessageBox. The send port uses the PassThruTransmit pipeline, which performs no processing of the message.
5. The message is delivered to the back end WCF service by the WCF-NetTcp send adapter.

Orchestration test scenario



1. The Visual Studio 2010 Ultimate edition load testing functionality generates an XML message and sends it to the computer running BizTalk Server using the NetTcp transport.
2. The XML message is received by a BizTalk Server receive port that uses the WCF-NetTcp receive adapter. The BizTalk Server receive port is configured to use the PassThruReceive, pipeline which performs no processing of the message.
3. The message is delivered to a simple Orchestration, which consists only of a receive port (bound to the WCF receive port from step 2) and a send port (bound to the WCF send port from step 4). Message variables are “un-typed”, meaning that they use a “Message Type” of “System.XML.XmlDocument”. The orchestration just receives the message through its receive port and sends the message through its send port. No message processing is performed.
4. A one-way BizTalk Server send port that uses the WCF-NetTcp send adapter subscribes to messages published by the orchestration and retrieves the message from the BizTalk MessageBox. The send port uses the PassThruTransmit pipeline, which performs no processing of the message.
5. The message is delivered to the back end WCF service by the WCF-NetTcp send adapter.

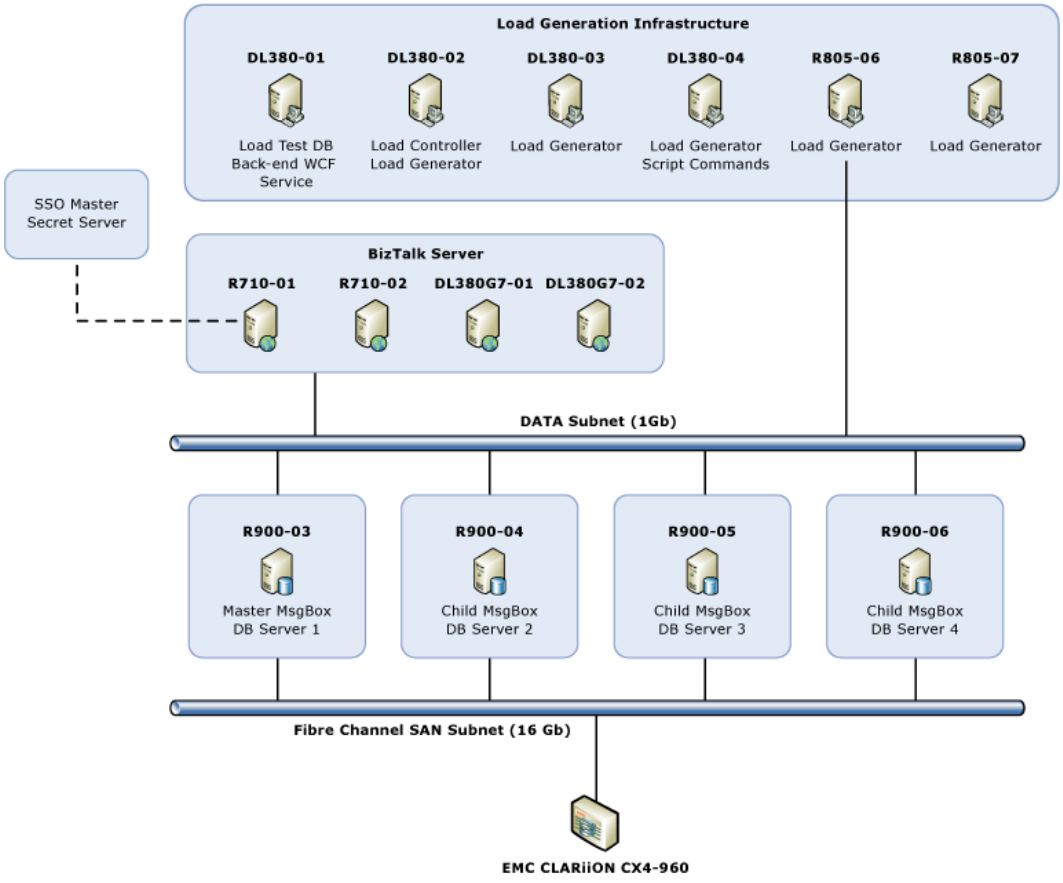
Hardware configuration

Lab hardware diagram and specifications

The hardware configuration used for the lab is illustrated below. To easily accommodate scale out of the processing and database tiers, the following lab hardware was used:

- Two Enterprise class Hewlett Packard DL-380 computers and two Enterprise class Dell R710 computers for the BizTalk Server processing tier.
- Four Enterprise class Dell R900 computers for the database tier, to provide multi-MessageBox capability.

A diagram of the hardware used in the lab is provided below:



The following table provides more specific information about the hardware used in the lab.

Name	Model	CPU type	Number of CPUs	Number of cores/CPU	CPU architecture	Memory	Operating System	Software
R710-01	Dell PowerEdge R710	Intel Xeon X5570	2 x 2.93 GHz	4	x64	72 GB	Windows Server 2008 R2 Enterprise Edition	BizTalk Server 2010
R710-02	Dell PowerEdge R710	Intel Xeon X5570	2 x 2.93 GHz	4	x64	72 GB	Windows Server 2008 R2 Enterprise Edition	BizTalk Server 2010

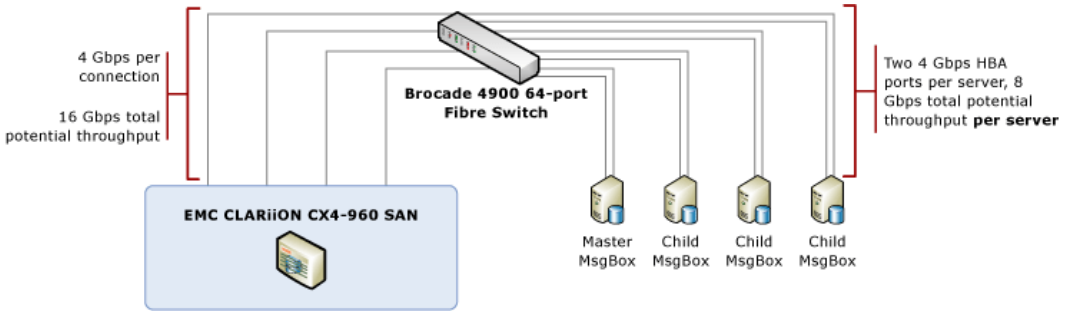
Name	Model	CPU type	Number of CPUs	Number of cores/CPU	CPU architecture	Memory	Operating System	Software
DL380G7-01	Hewlett Packard DL380 G7	Intel Xeon X5670	2 x 2.93 GHz	6	x64	192 GB	Windows Server 2008 R2 Enterprise Edition	BizTalk Server 2010
DL380G7-02	Hewlett Packard DL380 G7	Intel Xeon X5670	2 x 2.93 GHz	6	x64	192 GB	Windows Server 2008 R2 Enterprise Edition	BizTalk Server 2010
DL380-01	Hewlett Packard DL380	Intel Xeon 5150	2 x 2.66 GHz	2	x64	8 GB	Windows Server 2008 R2 Enterprise Edition	SQL Server 2008 R2 Load Test DB Visual Studio 2010 WCF back-end service
DL380-02	Hewlett Packard DL380	Intel Xeon E5335	2 x 2.00 GHz	4	x64	8 GB	Windows Server 2008 R2 Enterprise Edition	Visual Studio 2010 Load Test Controller
DL380-03	Hewlett Packard DL380	Intel Xeon E5335	2 x 2.00 GHz	4	x64	8 GB	Windows Server 2008 R2 Enterprise Edition	Visual Studio 2010 Load Test Agent
DL380-04	Hewlett Packard DL380	Intel Xeon E5335	2 x 2.00 GHz	4	x64	8 GB	Windows Server 2008 R2 Enterpris	Visual Studio 2010 Load Test

Name	Model	CPU type	Number of CPUs	Number of cores/CPU	CPU architecture	Memory	Operating System	Software
							e Edition	Agent. Perfmon Command-line
R805-06	Dell PowerEdge R805	AMD Quad-Core Opteron 2354	2 x 2.2 GHz	4	x64	32 GB	Windows Server 2008 R2 Enterprise Edition	Visual Studio 2010 Load Test Agent
R805-07	Dell PowerEdge R805	AMD Quad-Core Opteron 2354	2 x 2.2 GHz	4	x64	32 GB	Windows Server 2008 R2 Enterprise Edition	Visual Studio 2010 Load Test Agent
R900-03	Dell PowerEdge R900	Intel Xeon E7330	4 x 2.4 GHz	4	x64	64 GB	Windows Server 2008 R2 Enterprise Edition	SQL Server 2008 R2 with Cumulative Update 4
R900-04	Dell PowerEdge R900	Intel Xeon E7330	4 x 2.4 GHz	4	x64	64 GB	Windows Server 2008 R2 Enterprise Edition	SQL Server 2008 R2 with Cumulative Update 4
R900-05	Dell PowerEdge R900	Intel Xeon E7330	4 x 2.4 GHz	4	x64	64 GB	Windows Server 2008 R2 Enterprise Edition	SQL Server 2008 R2 with Cumulative Update 4

Name	Model	CPU type	Number of CPUs	Number of cores/CPU	CPU architecture	Memory	Operating System	Software
R900-06	Dell PowerEdge R900	Intel Xeon E7330	4 x 2.4 GHz	4	x64	64 GB	Windows Server 2008 R2 Enterprise Edition	SQL Server 2008 R2 with Cumulative Update 4

Storage area network configuration

The following diagram depicts the storage area network (SAN) configuration used for the lab environment.



EMC CLARiiON CX4-960 SAN configuration

Service processor and Cache information	LUN configuration
Two Service Processors, each with: • Read cache size: 2000 MB. • Write cache size: 8000 MB. • Two front-end ports attached to the fiber switch. 4 Gbps per port.	• 64 disks (268 GB, 15k RPM, Fiber Channel, Seagate. • Eight 8-disk RAID 1+0 Groups carved from these 64 disks. • Each DB server was assigned 5 MetaLUNs configured evenly from these RAID Groups.

It is important to determine the maximum attainable throughput of a SAN and compare this value to the expected load against the SAN in production. This will help prevent unanticipated expenditures for SAN-related hardware when moving your application from a test/quality assurance (QA) environment into a production environment. For example, the maximum available throughput for the SAN may be more than sufficient for your application in a sandboxed test

environment. However, if other server applications will use the SAN in production, available SAN throughput may be insufficient and could become a bottleneck.

When evaluating performance of your SAN, consider the following:

1. **What is the maximum attainable throughput of the SAN?** – This is measured using the lowest common denominator, in terms of throughput, of the host bus adapter (HBA) in the server, the Switch-SAN throughput, and the speed of the SAN controller cards.
2. **What other applications will use the SAN in production?** – Consider what other applications will make use of the SAN in the production environment. If other database or otherwise I/O intensive applications such as Exchange Server will use the SAN in production, the amount of SAN throughput available for your computer running BizTalk Server will be reduced accordingly.

For the lab environment, a dedicated SAN was used. Each of the four SQL Server computers was connected to the SAN switch with two 4-Gbps host bus adapters (HBAs), providing a total potential throughput of 8-Gbps per server. The SAN had two service processors, and each service processor had two front-end ports attached to the fiber switch, providing a total potential throughput of 16-Gbps between the SAN and the switch.

The following LUN configuration was used for each of the four computers running SQL Server in the test environment.

Drive letter	Volume name	Files	LUN size (GB)
C	Local C drive	MASTER,MSDB, and MODEL	136
H	Data_Tempdb	TempDB data files	50
I	Logs_Tempdb	TempDB log files	50
J	Data_BtsMsgBox	BizTalk MsgBoxDB data files + (on the Master MsgBox) Mgmt DB, SSO DB, DTA DB data files	120
K	Logs_BtsMsgBox	BizTalk MsgBoxDB log files + (on the Master MsgBox) Mgmt DB, SSO DB, DTA DB log files	120
O	Logs_Spare	Not used for SQL files. Used to store DTCLog file, as MSDTC cause frequent disk I/O, especially in a multi-MessageBox environment.	20

SQLIO test results

We used the SQLIO tool to benchmark and measure the Input/Output capacity of the storage area network (SAN) configuration used in our lab environment. As the name of the tool implies, SQLIO is a valuable tool for measuring the impact of file system I/O on SQL Server performance. SQLIO can be downloaded from http://go.microsoft.com/fwlink/?LinkID=210381.
To measure Input/Output capacity of the storage area network (SAN) configuration for SQL Server database applications, see Analyzing I/O Characteristics and Sizing Storage Systems for SQL Server Database Applications (http://go.microsoft.com/fwlink/?LinkID=210379).

For our SQLIO test, the sqlio.exe utility issues 8K read requests from 8 threads and maintain an I/O queue depth of 8. We used the following parameters:

- All tests used 8 processing threads, each running for 60 seconds.
- 8kb random writes to the data files.
- 8kb random reads to the data files.
- All files are sized at 25 GB.
- The disk drives to be benchmarked are H:, I:, J:, and K drive.

The SQLIO command lines used are follows:

- **For reads**: sqlio -kR -s60 -frandom -o8 -t8 -b8 -LS -FParam.txt
- **For writes**: sqlio -kW -s60 -frandom -o8 -t8 -b8 -LS -FParam.txt

The Param.txt file contains the following:

- **Number of drives:** H:,I:,J:, and K
- **Physical location of the testing files**:
  - H:\testfile.dat 2 0x0 25000
  - I:\testfile.dat 2 0x0 25000
  - J:\testfile.dat 2 0x0 25000
  - K:\testfile.dat 2 0x0 25000

The following table lists the test results:

- 8 KB random reads from 25 GB test files simultaneously on LUNs H:, I:, J:, K: (the LUNs used for all of our database files)

Input/Output operations per second (IOs/sec)	Megabytes per second (MBs/Sec)	Average Latency
10662.67	83.30	5 ms

- 8 KB random writes to 25 GB test files simultaneously on LUNs H:, I:, J:, K: (the LUNs used for all of our database files)

Input/Output operations per second (IOs/sec)	Megabytes per second (MBs/Sec)	Average Latency
31527.95	246.31	1 ms

See Also

Scaling a Production BizTalk Server Environment

Observations and Recommendations

Test Results Summary

The results for the Messaging-only scenario were observed to be ~ 109,382,400 messages per day. For the Orchestration scenario, the results indicate that a single computer running BizTalk Server can process up to 58,752,000 messages per day. These results were achieved in a sandboxed environment using normal enterprise-class hardware deployed for many BizTalk Server solutions. Therefore, these results indicate the type of BizTalk Server performance of that can be achieved with no custom code. Customer solutions often involve custom-developed BizTalk artifacts; in many cases this increases processing requirements which in turn affects performance. By following the advice presented in this guide, particularly the Optimizing BizTalk Server Applications section, the impact of implementing custom-developed BizTalk Server artifacts can be minimized.

The following table provides a summary of the test results for this performance assessment.

Scenario	Messaging (BizTalk KPI – Documents Processed per Second)	Messaging (SQL KPI – SQL Processor Utilization)	Messaging (Latency)	Orchestration (BizTalk KPI – Documents Processed per Second)	Orchestration (SQL KPI – SQL Processor Utilization)	Orchestration (Latency)
Single MessageBox 1 BizTalk Server computer	1266 documents processed per second	59% SQL CPU utilization	0.06	680 documents processed per second	66.5% SQL CPU utilization	0.067
Single MessageBox 2	1267 document	59.8% SQL CPU	0.057	686 documents	68.5% SQL CPU	0.067

Scenario	Messaging (BizTalk KPI – Documents Processed per Second)	Messaging (SQL KPI – SQL Processor Utilization)	Messaging (Latency)	Orchestration (BizTalk KPI – Documents Processed per Second)	Orchestration (SQL KPI – SQL Processor Utilization)	Orchestration (Latency)
BizTalk Server computers	s processed per second	utilization		processed per second	utilization	
3 MessageBox 1 BizTalk Server computers	2102 documents processed per second	41% SQL CPU utilization	0.077	974 documents processed per second	48% SQL CPU utilization	0.11
3 MessageBox 2 BizTalk Server computers	2285 documents processed per second	58% SQL CPU utilization	0.041	1065 documents processed per second	65% SQL CPU utilization	0..69
4 MessageBoxes 1 BizTalk Server computers	2125 documents processed per second	30% SQL CPU utilization	0.078	979 documents processed per second	37% SQL CPU utilization	0.095
4 MessageBoxes 2 BizTalk Server computers	2790 documents processed per second	50% SQL CPU utilization	0.052	1487 documents processed per second	63% SQL CPU utilization	0.15
4 MessageBoxes 3 BizTalk Server	2656 documents processed	58% SQL CPU utilization	0.074	1388 documents processed per second	65% SQL CPU utilization	0.15

Scenario	Messaging (BizTalk KPI – Documents Processed per Second)	Messaging (SQL KPI – SQL Processor Utilization)	Messaging (Latency)	Orchestration (BizTalk KPI – Documents Processed per Second)	Orchestration (SQL KPI – SQL Processor Utilization)	Orchestration (Latency)
computers	per second					

Comparison of performance statistics with BizTalk Server 2009

The following table shows a comparison of performance statistics between BizTalk Server 2009 and BizTalk Server 2010. The BizTalk Server 2009 performance statistics listed in this table are from the BizTalk Server 2009 Performance and Optimization Guide.

Scenario	BizTalk Server 2009 Maximum Sustainable Throughput (MST) msgs/sec	BizTalk Server 2009 Number of BizTalk Servers needed	BizTalk Server 2010 Maximum Sustainable Throughput (MST) msgs/sec	BizTalk Server 2010 Number of BizTalk Servers needed	BizTalk Server 2010 % difference from BizTalk Server 2009
Messaging - Single MessageBox	1291	2	1266	1	98.06%
Orchestration - Single MessageBox	676	2	680	1	100.59%
Messaging - 3 MessageBoxes	2103	3	2285	2 (2102/sec with 1 BTS)	108.65%
Messaging - 4 MessageBoxes	Not applicable	Not applicable	2790	2	Not applicable
Orchestration - 3 MessageBoxes	1005	4	1065	2 (974/sec with 1 BTS)	105.97%
Orchestration - 4 MessageBoxes	Not applicable	Not applicable	1487	2	Not applicable

In our lab environment, when using four MessageBox databases, the maximum sustainable throughput results were as follows:

- For messaging scenario, 2790 documents per second.
- For orchestration scenario, 1487 documents per second.

Message considerations While BizTalk Server imposes no restriction on message size, the tests we ran for BizTalk Server 2010 used 2-KB messages only and the type of WCF adapter used was WCF-NetTCP adapter. This matches the message size and adapter type used in the testing for the BizTalk Server 2009 Performance Optimization Guide.

The drivers for the performance improvement are:

1. **Hardware advancements** - The SQL Server computers used in our lab were 4-CPU, quad-core (16 cores), Intel Xeon E7330 @ 2.40 GHz. In the 2009 tests, 4-CPU, quad-core (16 cores), Intel Xeon 2.4 GHz were used.

   The BizTalk Server computers used in our lab were 2-CPU, quad-core (8 cores), Nehalem Hyper-Threaded (16 logical cores), Intel Xeon X5570 @ 2.93 GHz. In the 2009 tests, 2-CPU, quad-core (8 cores), Intel Xeon 2.33 GHz were used.
2. **Improvement of SQL Server Engine** - The tests in 2009 were performed against SQL Server 2008, whereas our tests were performed with SQL Server 2008 R2 as the underlying database platform.

Recommendations for scaling the BizTalk Server and SQL Server tiers

With these results, the BizTalk Server Product Team was able to demonstrate that a single BizTalk Server computer and a single SQL Server computer can support over 109 million messages in a messaging scenario and 58 million orchestrations during a 24-hour period. By scaling the BizTalk Server and SQL tiers to the optimal configuration available in our environment, we were able to process over241 million messages per day and over 128 million orchestrations. The results were performed in a sandboxed environment by using the class of hardware deployed in many enterprises. As stated earlier, these numbers represent realistic performance of BizTalk Server that can be achieved with no custom code and an optimized environment. With additional hardware, it is possible that even greater performance could have been achieved. Customer solutions often involve custom-developed BizTalk artifacts that incur additional processing requirements, hence increasing resource utilization and in turn reducing overall performance. However, by following the advice outlined throughout the guide, particularly the recommendations in Optimizing BizTalk Server Applications, the impact of this can be minimized.

The results demonstrate that scaling out the number of BizTalk Server computers is an effective scale-out strategy if the MessageBox SQL Server computer is not a bottleneck. After a certain point our results indicate that adding BizTalk Server computers becomes an ineffective scale-out technique because we observed that a contention point occurred on the shared tables within the MessageBox database, which caused performance to diminish. To maximize the results that can be obtained from a BizTalk Server group with a single MessageBox database, you should apply the optimizations described in Optimizing Database Performance. In particular, you should ensure that a fast storage subsystem is used for the SQL Server storage and that the logical

disks used by SQL Server to store the MessageBox database files respond in the shortest possible time. A commonly used threshold for measuring acceptable read/write performance is 15 milliseconds; typically this is measured by the Avg. Disk sec/Read and Avg. Disk sec/Write counters that can be found under the Logical Disk performance object. Once all available optimizations have been applied to the SQL Server computer that hosts the MessageBox database, additional MessageBox databases can be added. The same optimization techniques that were applied to the primary MessageBox database should also be applied to the secondary databases. We recommend that you follow the guidelines in Scaling Out the SQL Server Tier (http://go.microsoft.com/fwlink/?LinkID=158075) in the BizTalk Server documentation.

To gain optimal results, the whole hardware and software stack needs to be of appropriate quality and also configured correctly. First, good quality hardware should be purchased including, but not limited to, gigabit networking, fast storage (SAN or 15-K local SQL disks) and modern computers that have multiple CPUs with multiple cores per CPU. The SQL Server computer should be dedicated to BizTalk Server processing only. When running a single SQL Server computer, we recommend that two instances of SQL be created, one for the BizTalk MessageBox and one for all other databases. This enables instance-wide settings to be configured for optimal performance of the MessageBox. The optimizations recommended in Optimizing Performance should be applied step by step in the following order: Optimizing Operating System Performance, Optimizing Network Performance, Pre-Configuration Database Optimizations, Post-Configuration Database Optimizations and General BizTalk Server Optimizations. Creating dedicated filegroups for the MessageBox database and allocating these across the SAN LUNs, as described in Optimizing Filegroups for the Databases, can provide considerable performance improvement depending on your SAN configuration and LUN layout.

To effectively determine how to scale the BizTalk Server or SQL tier, we recommend that load testing be performed using messages that approximate actual production data. Before scaling the BizTalk Server tier, ensure that SQL Server is not already a bottleneck, as recommended in Monitoring SQL Server Performance. If SQL Server is not a bottleneck and there is CPU headroom on any of the BizTalk Server computers, you may be able to improve throughput by modifying your host instance layout. It is important to establish four or five key performance indicators (KPIs), which are used as high-level comparison points for all test runs. Following this advice, you will be able to quickly gauge whether a particular change degraded overall performance of the solution.

Before scaling out the SQL Server tier, apply all of the optimizations in Optimizing Database Performance. During the course of customer performance labs, the observation was that disk storage configuration for the MessageBox and TempDb databases in particular can provide over 30 percent throughput improvement. When scaling to multiple MessageBox databases, three and four MessageBox databases were used because there is little performance advantage when scaling out from a single MessageBox database to two MessageBox databases. For more information about scaling out the BizTalk Server MessageBox, see Scaling Out the SQL Server Tier (http://go.microsoft.com/fwlink/?LinkID=158075) in the BizTalk Server documentation.

Implemented Optimizations

This section provides a checklist of all optimizations that were applied for the lab test scenarios.

Note

These should be tested in accordance with your change management procedures before applying these within your production environment.

Applying the optimizations in a systematic, controlled manner—by applying a set of optimizations, then testing the impact—will result in the maximum performance benefit. Applying optimizations without periodically testing the impact of the optimizations may actually result in a performance degradation of the solution.

Checklists of Optimizations Applied

Platform and Network Optimizations

Optimization	Reference
BIOS: Configure settings for performance.	Optimizing Operating System Performance
Disable real-time scanning on SQL Server files.	Optimizing Operating System Performance
Disable Antivirus on all BizTalk Server and SQL Server computers.	General Guidelines for Improving Operating System Performance

SQL Server Optimizations: General

Optimization	Reference
Set NTFS File Allocation Unit to 64 KB.	Pre-Configuration Database Optimizations
Install SQL Server 2008 R2.	Pre-Configuration Database Optimizations
Configure SQL Server 2008 R2 Data Collector/Warehouse.	Pre-Configuration Database Optimizations
Ensure that appropriate Windows privileges have been extended to the SQL Server Service accounts.	SQL Server 2008 R2: Setting Up Windows Service Accounts (http://go.microsoft.com/fwlink/?LinkID=132144)
Set Minimum and Maximum Server Memory for SQL Server.	Pre-Configuration Database Optimizations
Grant the Windows Lock Pages In Memory privilege to the account that is used for SQL Server.	Pre-Configuration Database Optimizations
Pre-size BizTalk Server databases to appropriate size with multiple data files..	Post-Configuration Database Optimizations
Configure SQL Server Client Protocols.	Troubleshooting SQL Server (http://go.microsoft.com/fwlink/?LinkID=154250)

Optimization	Reference
Split the tempdb database into multiple data files of equal size on each SQL Server instance used by BizTalk Server	Pre-Configuration Database Optimizations
Manually set SQL Server Process Affinity	Pre-Configuration Database Optimizations
Configure MSDTC and disable DTC tracing	Pre-Configuration Database Optimizations
Enable Trace Flag T1118 as a startup parameter for all instances of SQL Server	Pre-Configuration Database Optimizations

SQL Server Optimizations: BizTalk Databases

Optimization	Reference
Set Autogrowth of BizTalk databases to a fixed value and not a percentage value.	Post-Configuration Database Optimizations
Move/split BizTalk database data and log files to separate LUNS.	Post-Configuration Database Optimizations
Consider setting the 'text in row' table option on specific MessageBox database tables	Post-Configuration Database Optimizations

BizTalk Optimizations

Optimization	Reference
Separate receive ports, send ports, orchestrations, and tracking on separate, dedicated hosts.	General BizTalk Server Optimizations
Configure polling intervals.	Low-Latency Scenario Optimizations
Adjust the BizTalk configuration file maximum connection property.	“Increase the number of SOAP and HTTP concurrent connections allowed by changing the value for the maxconnection parameter” section of General BizTalk Server Optimizations
Define CLR Hosting parameters for each host instance on each BizTalk Server node: Maximum IO Threads: 250 Maximum Worker Threads: 100	“Define CLR hosting thread values for BizTalk host instances” section of General BizTalk Server Optimizations

Optimization	Reference
Minimum IO Threads: 25 Minimum Worker Threads: 25	
Increase In-process messages and Internal Message Queue Size to 10000.	Low-Latency Scenario Optimizations
Disable BizTalk Server Group-level tracking.	General BizTalk Server Optimizations
Manage the number of concurrently executing requests for ASP.NET 4 Web applications that can host isolated received locations, back-end Web services and WCF services on IIS 7.5 and IIS 7.0 running in Integrated mode	General BizTalk Server Optimizations
Disable BizTalk Server host throttling	General BizTalk Server Optimizations
Configure MSDTC and disable DTC tracing	General BizTalk Server Optimizations

WCF Configuration Optimizations

Optimization	Reference
For each WCF service, apply the serviceThrottling service behavior, and set **maxConcurrentCalls** and **maxConcurrentSessions** to 200.	Optimizing BizTalk Server WCF Adapter Performance
Configure the usage of serviceThrottling behavior in the backend WCF service configuration file.	Optimizing WCF Web Service Performance

See Also

Scaling a Production BizTalk Server Environment

Key Performance Indicators

This topic provides test results that the BizTalk Server product group observed when using the following scale-out methods:

- Key performance indicators (KPIs) when increasing the number of BizTalk Server computers in a BizTalk Server group. For these tests, only one BizTalk Server MessageBox database was configured for the BizTalk Server group. These tests focused solely on the impact of adding more BizTalk Server computers to a BizTalk Server group.

- KPIs when increasing the number of BizTalk Server MessageBox databases used by the BizTalk Server group. These tests focused solely on the impact of adding more BizTalk Server MessageBox databases to a BizTalk Server group.
- KPIs when increasing the number of both BizTalk Server computers and BizTalk Server MessageBox databases used by the BizTalk Server group. These tests measured the impact of adding both BizTalk Server computers and BizTalk Server MessageBox databases to a BizTalk Server group.

Analysis of key performance indicators

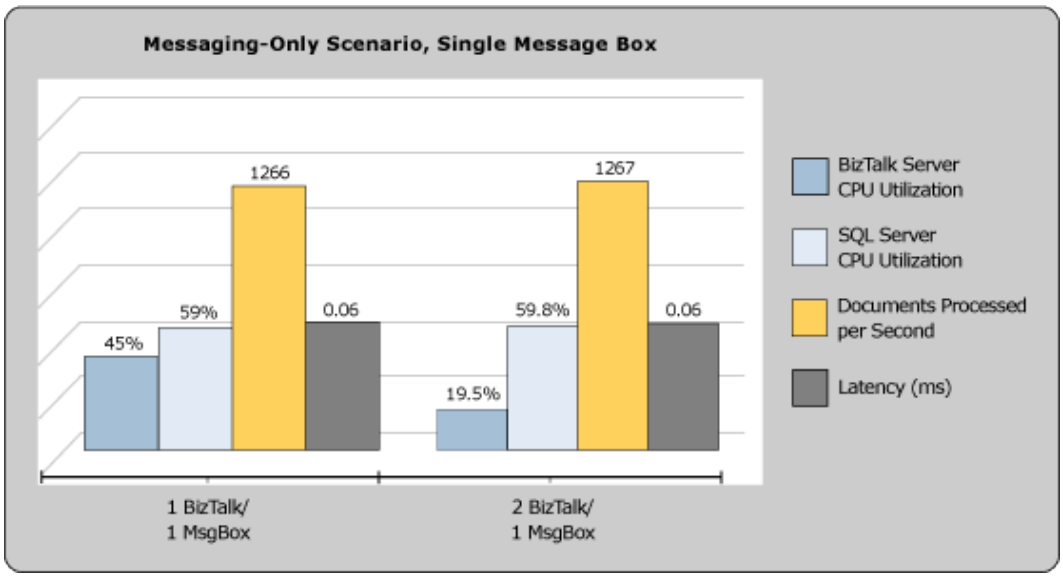
Messaging scenario, BizTalk Server scale-out – BizTalk and SQL KPI

Adding a second computer running BizTalk Server does not show a significant impact on the overall throughput. The load on the BizTalk Server CPU decreases by 25%. The CPU for SQL Server marginally increases from 59% to 59.8% when the second computer running BizTalk Server is added to the BizTalk Server group. Beyond this point, no further performance benefit was gained by increasing the number of BizTalk processing servers.

Each BizTalk host instance regularly polls the appropriate queue in the MessageBox. Any message that is referenced on the host queue is actually stored within the shared set of tables in the MessageBox. If throughput drops when adding more computers running BizTalk Server, a common cause is too much activity against the shared tables within the MessageBox database. A dedicated I/O path for SQL Server to these tables can be created by assigning these tables to a specific filegroup.

Optimizing Filegroups for the Databases provides guidance on how to assign tables to specific filegroups. The script included in BizTalk Server MessageBox Database Filegroups SQL Script of the guide tells how you can accomplish this. Scaling out to a multiple MessageBox configuration should only be considered after distributing MessageBox objects across multiple filegroups, and after all other SQL-related optimizations have been applied.

Percentage of BizTalk Server and SQL Server CPU utilization

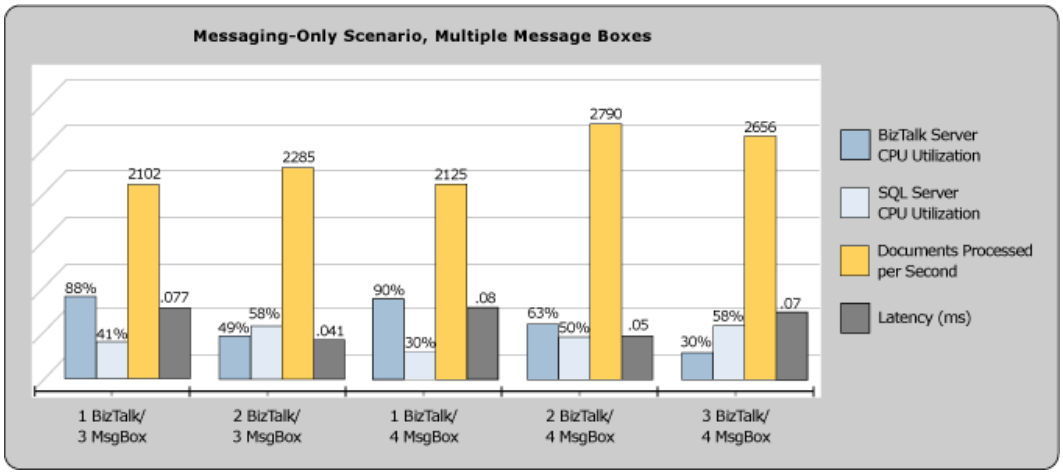


Messaging scenario, BizTalk Server and SQL Server scale-out – BizTalk and SQL KPI

This test was performed to determine the effectiveness of scaling out the SQL Server tier by adding four MessageBox databases. In this scenario, adding up to two computers running BizTalk Server enabled a maximum sustainable throughput of 2,790 messages per second. This was 118% higher than the maximum obtainable throughput when using only a single MessageBox. Beyond this point, adding more processing power to the BizTalk Server tier degraded performance in a similar fashion to the single MessageBox scenario.

The key findings from the Messaging scenario tests are that scaling out BizTalk Server is an effective technique to increase overall throughput if contention on SQL Server is not a bottleneck. If the MessageBox database becomes a contention point, first apply the optimizations detailed in Optimizing Database Performance, particularly the filegroup optimization script described in BizTalk Server MessageBox Database Filegroups SQL Script to distribute I/O load. If you are still unable to achieve your desired throughput, you should consider scaling out by adding more MessageBox databases.

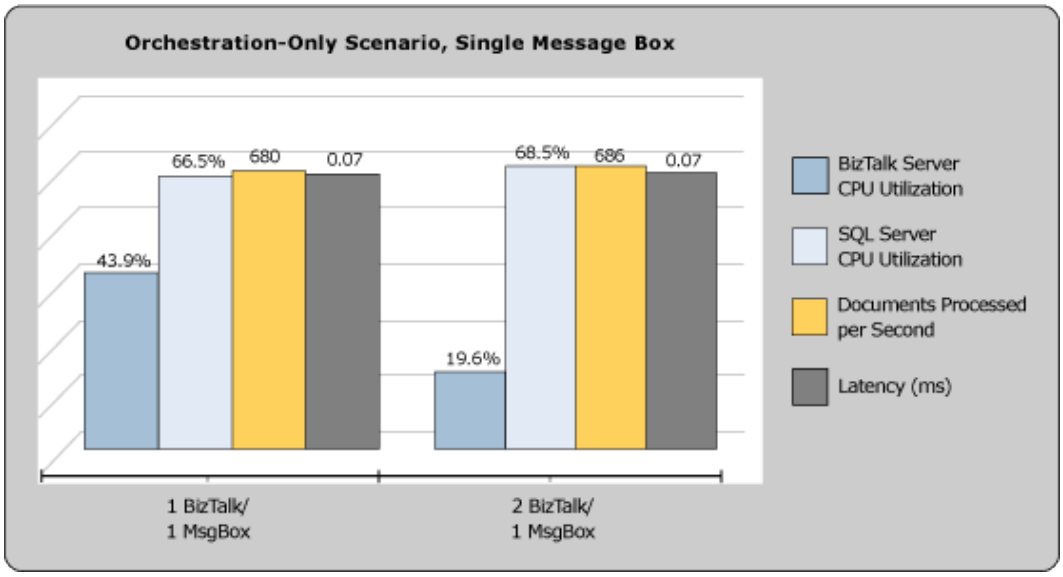
Percentage of BizTalk Server and SQL Server CPU utilization



Orchestration scenario, BizTalk Server scale-out – SQL Server and BizTalk Server KPI

Adding a second computer running BizTalk Server does not show a significant impact on the overall throughput. The load on the BizTalk Server CPU decreases by 23 percent. The CPU for SQL Server increases from 66.5 percent to 68.5 percent when an additional computer running BizTalk Server is added.

Percentage of BizTalk Server and SQL Server CPU utilization

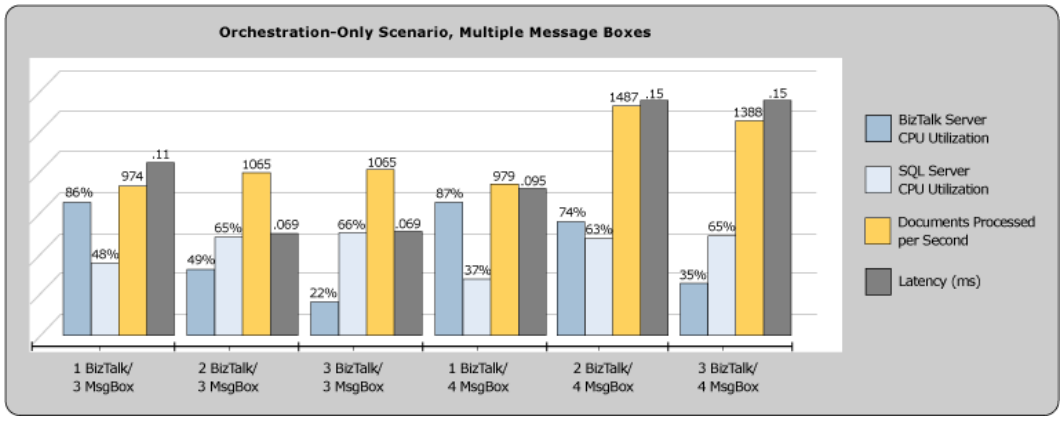


Orchestration scenario, BizTalk Server and SQL Server scale-out – SQL Server and BizTalk Server KPI

This test was performed to determine the effectiveness of scaling out both the BizTalk Server and SQL Server tier by adding more computers running BizTalk Server and more MessageBox databases for the Orchestration scenario. In this scenario, adding up to two computers running BizTalk Server enabled a maximum sustainable throughput of 1,487 orchestrations per second.

This was 116% higher than the maximum obtainable result against a single MessageBox. Scaling out to four MessageBox databases on separate SQL Server computers accommodates increased throughput due to additional processing power and the ability to distribute database load across multiple MessageBox databases. This tactic also relieves contention on shared tables, which was a bottleneck in the single MessageBox environment. As with the Messaging scenario, increasing the number of MessageBox databases and distributing these across dedicated SQL instances enables the addition of several BizTalk Server computers to the BizTalk Server group.

Percentage of BizTalk Server and SQL Server CPU utilization



See Also

Scaling a Production BizTalk Server Environment

BizTalk Server Performance Testing Methodology

This section provides information about the performance testing methodology used for this guide.

In This Section

- Establishing Performance Criteria
- Phases of a Performance Assessment
- Implementing Automated Testing
- Unit Testing

Establishing Performance Criteria

The performance goals of a BizTalk Server solution typically fall into one of two categories, throughput or latency. This topic describes the factors that should be considered when evaluating the throughput or latency of a BizTalk Server solution.

Note

Optimizing throughput or latency of a BizTalk Server solution often represents conflicting goals. For example, you might improve the throughput of the file receive adapter by increasing the batch size, but doing so would reduce latency. In this scenario the adapter takes longer to accumulate messages for a larger batch size, which in turn will reduce end-to-end latency for a given message.

Factors affecting throughput of a BizTalk Server solution

Throughput- Broadly measured as the number of documents that a BizTalk Server solution can process within a given time interval.

Factors that affect throughput include:

- **Message size** – Larger messages consume more overhead than smaller messages, especially if the messages are transformed with a map and are large enough so that they are buffered to the file system during the mapping operation. For more information about how message size affects BizTalk Server performance, see How BizTalk Server Processes Large Messages (http://go.microsoft.com/fwlink/?LinkId=139293).
- **Message format** - Messages are received into BizTalk Server in one of two primary formats, XML files or flat files. Because flat files must be translated into the XML format to be processed by the BizTalk Messaging engine, additional overhead is incurred by the processing of flat files.
- **Adapter requirements** – Different adapters frequently have varying performance capabilities. For example, adapters that require MSDTC transaction support will incur additional overhead/reduced performance as compared to an adapter that does not use MSDTC transactions. Adapters used by the BizTalk solution will vary depending on your trading partner's requirements and/or internal business needs.
- **Orchestration processing requirements** – Orchestrations provide great flexibility for encapsulating and applying business processes to messages received by BizTalk. At the same time, orchestrations consume overhead, which must be considered when estimating the throughput of a BizTalk Server solution.
- **Peak load requirements** – Most document processing does not necessarily occur in a measured orderly fashion. For example, a BizTalk Server solution may sustain a large percentage of its processing load at the close of a business day. Therefore peak load requirements and the Maximum Sustainable Throughput (MST) of a BizTalk Server solution should be taken into account when establishing throughput criteria. For more information about measuring the MST of a BizTalk Server solution, see Measuring Maximum Sustainable Engine Throughput (http://go.microsoft.com/fwlink/?LinkID=154388) and Measuring Maximum Sustainable Tracking Throughput (http://go.microsoft.com/fwlink/?LinkID=153815).
- **Document tracking requirements** – Document tracking imposes additional overhead on the system. Document tracking requirements should be a primary consideration when estimating the throughput goals of a BizTalk solution.

See Also

BizTalk Server Performance Testing Methodology

Phases of a Performance Assessment

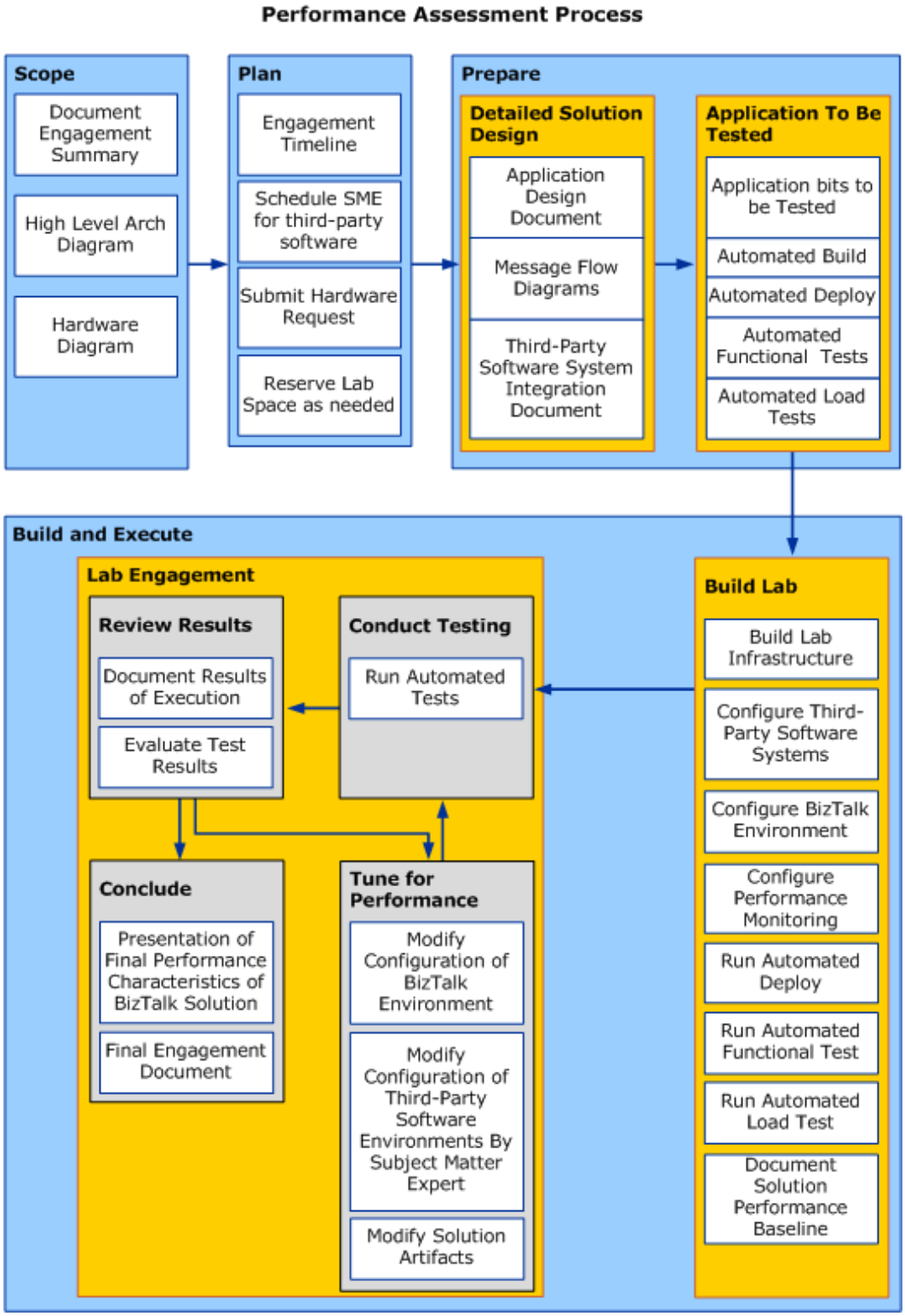
One of the primary goals of BizTalk Server is to provide maximum flexibility for accommodating as many processing scenarios as possible. Because of this great flexibility, one of the primary challenges facing developers of a BizTalk solution is to determine how to make best use of the features available in BizTalk Server to meet their business needs. This flexibility also poses a challenge when optimizing the performance of a BizTalk Server solution.

This section describes the methodology that should be used to optimize the performance of a BizTalk Server solution by engaging in a BizTalk Server performance assessment. A BizTalk Server performance assessment is used to maximize the performance of a particular BizTalk Server solution. In order to gain the maximum benefit from a BizTalk Server performance assessment, the performance assessment should be divided into the following five distinct steps/phases:

1. Scope
2. Plan
3. Prepare
4. Build lab
5. Execute

This topic describes each of these phases in greater detail.

Phases of a BizTalk Server performance assessment



In This Section

- Phase 1: Scoping the Assessment - Describes the steps that should be followed when establishing the scope of the performance assessment.
- Phase 2: Planning the Assessment – Describes how to create a solution milestones timeline. This is used as a master plan to clearly document when expected goals for the testing of the solution should be met.
- Phase 3: Preparing for the Assessment – Describes how to provide a detailed architectural diagram of the solution and specific aspects of the hardware configuration used by the solution.
- Phase 4: Building the Assessment Environment – Describes installation of hardware and applications according to the detailed design of the solution platform that was established previously.
- Phase 5: Executing the Assessment – Provides information about generating performance data via automated load testing and how to detect and eliminate any bottlenecks in the solution.

See Also

BizTalk Server Performance Testing Methodology

Phase 1: Scoping the Assessment

This topic describes the aspects of the scope phase of a BizTalk Server performance assessment.

A common mistake when engaging in a performance assessment is to underestimate the scope of the performance assessment. If sufficient resources and time are not allocated beforehand, then the team responsible for the performance assessment will be unable to complete all of the tasks required to build out and test a complex production-like environment. The team working on the performance assessment should choose their battles carefully. Most performance assessments are performed within a very limited timeframe and so the team should identify and focus on one or two, maybe three at most, key performance objectives. The application to be tested should be specifically developed to focus on the identified performance objectives and should factor out all other technology variables as much as possible. This topic describes the aspects of the scope phase of a BizTalk Server performance assessment.

Considerations before beginning the performance assessment

The following factors should be considered before any other work is done for a performance assessment. These factors will help decide the general direction of the scope phase and are a good starting point for a performance assessment.

Message load

It’s important to consider right from the outset how you are going to replicate the message load that will actually be going through the production system. For instance, if in production 20 percent of the messages will be <20KB in size, 50 percent will be <100KB in size and the remaining 30 percent could be up to 1 MB in size, it is important that this be replicated in the lab.

Develop a brief detail of the scenarios to be tested

After you have identified the test cases that will be tested, it is important to identify the major components that are involved in them. This includes both BizTalk Server components (such as messaging and orchestrations) and other components, including third-party technologies such as MQSeries or SAP. It is very important that you be aware of all of these from the outset as it will help you gauge the complexity of the lab and will enable you to plan for the technical skills required during the engagement.

Identify which adapters will be used

It is of vital importance the performance assessment lab use the actual adapters that will be used in the production environment. Failure to use the actual adapters used in production causes problems because the performance of different adapters varies significantly. For instance, the File adapter is one of the fastest BizTalk Server adapters but it lacks some of the features required of some systems; in particular the File adapter does not provide support for transactions. Transaction support provides the ability to read messages and write them to the MessageBox database, all within the context of an MSDTC transaction. Transaction support incurs overhead. Therefore, using the File adapter to simulate a production scenario that requires transaction support would not be a viable comparison. In this scenario, the performance results are essentially invalid because they are very unlikely to reflect the actual performance of the production system.

Estimate time required for the performance assessment

Use the following guidelines to estimate the time required for the performance assessment:

- Allocate two weeks of preparation time for setting up the testing environment. One week will be used to build out the required infrastructure, software components, and apply software updates. The second week will be dedicated to setting up the specific environment that duplicates the BizTalk Server production environment.
- Allocate an additional week for each test case that will be analyzed during the performance assessment.
- Allocate a week at the end of the actual performance assessment to document the findings of the performance assessment.

So for a typical performance assessment with two test cases, the estimated time to complete the assessment would be:

(two weeks of preparation time) + (two weeks for the actual performance assessment) + (one week to document findings) = five weeks total.

Documenting the performance assessment

As part of scoping the performance assessment, the details of the assessment should be documented in an engagement summary. This document should clearly define goals and objectives, stakeholders, and milestones for the performance assessment. This document will serve as a roadmap for the performance assessment, help ensure all stakeholders agree on the engagement details and serve to ensure the performance assessment stays on track. This document should be updated throughout the performance assessment and include a summary of results upon the conclusion of the performance assessment.

Goals and objectives

Carefully define throughput and latency goals - What performance criteria are you trying to maximize? Typically one or more of the following performance measures are the focus of a BizTalk Server performance assessment.

- **Throughput** – Typically measured as the number of messages per time unit that can be processed over a specified time interval. For example, a throughput goal might be defined as the ability to process 100 messages per second for a period of 3 hours.
- **Latency** – Usually defined as percentage of all messages that can be processed end-to-end within a given time interval, for example:
  - 95 percent of all messages should be processed end-to-end within two seconds.
  - 100 percent of all messages should be processed end-to-end within four seconds.
- **Peak loads**
- **Time to complete a batch of** *X*
- **Floodgate recovery scenarios**
- **High-level architecture including hardware and software**

Performance goals should be measured in terms of "achieve highest possible *X* given hardware and software constraints." Determine how the goal will be measured in advance. For example, "Maximum Sustainable Throughput of *X* end-to-end as measured by the counter 'XLang/s\orchestrations completed / second'".

Note

In the examples above, *X* is placeholder for the unit that is the focus of the performance assessment. *X* could represent orchestrations, messages, or other performance metrics that are relevant to the BizTalk solution.

Is the desired performance attainable?

When evaluating performance considerations, you should first determine if the available hardware resources will be sufficient for meeting your performance goals. In some cases, the desired performance is simply not possible without additional investment in hardware. There is no magic formula that can be used to determine what performance is or is not possible given a specific hardware configuration as many factors influence the performance of the solution, including the complexity of orchestrations, custom code that is used, number of times that messages are persisted to the MessageBox database, and limitations of external systems. The table below provides a summary of hardware used to achieve specific performance goals for certain scenarios.

Note

The information below is provided for guidance only. The requirements of different BizTalk Server solution vary greatly so the values in this table should be used as a rough "rule of thumb" when estimating required hardware resources.

Sample hardware scenarios

Type of Processing	BizTalk Server computers	SQL Server computers	Throughput attained	Notes
Orchestration exposed as a Web service, MQSeries adapter	• 6 BizTalk Server 2010 computers • Each server running two 3-GHZ dual core processors • Windows Server 2008 R2, 8GB memory	• 3 SQL Server 2008 R2 computers • Each server running four 3-GHZ dual core processors • 64-bit, 16 GB memory • Single MessageBox database	95 orchestrations per second	• Average latency 1.11s • 99% messages processed with <2 seconds latency
Messaging	6 BizTalk Server 2010 computers	3 SQL Server 2008 R2 computers	Peak throughput attained over a 2 hour period was 277 messages per second	Before optimizing solution, peak throughput was 125 messages per second
Low-latency scenario using orchestrations and Web services	One dual processor BizTalk Server 2010 computer	One quad processor SQL Server 2008 R2 computer	60 orchestrations per second	< 350 millisecond latency achieved
Low-latency scenario using orchestrations	23 Quad processor BizTalk Server 2010 computers	• 3 SQL Server 2008 R2 computers • One 16-CPU master MessageBox database • Two 8-CPU secondary MessageBox database computers	1156 orchestrations per second	As of January 2010 was the highest sustainable performance of BizTalk Server running orchestrations recorded

After the available hardware infrastructure is deemed sufficient to achieve the desired performance, consider the following when scoping a BizTalk Server performance assessment:

- Which adapters and/or accelerators are required?
- What are the requirements for implementing orchestrations in the solution?
- Document throughput requirements: What are the maximum sustainable throughput requirements for the solution?
- Latency requirements: How responsive does the solution need to be for solicit-response and request-response scenarios?
- How well does the solution recover from periods of peak document load?
- What are the high availability requirements of the solution?
- What are the document tracking requirements of the solution?
- What are the performance characteristics of any dependent applications such as a remote Web service or other system? If dependent applications do not keep up with the required load, the overall system performance will be degraded accordingly.

Hardware information to consider during scope phase

When scoping the solution, create a high-level hardware diagram that includes the following:

- Computer architecture (such as x86, x64, and IA64)
- CPU requirements (such as type, speed, number, cores, and use of hyperthreading)
- RAM requirements for each computer
- Local disk storage (type, size, speed)
- SAN (storage requirements: number of LUNS; SAN card type)
- Network cards (number in each computer, 100 megabits per second (Mbps) versus 1 gigabit per second (1 Gbps).)
- How will firewalls be deployed in the solution?
- Will Network Load Balancing hardware be used?
- Are specific computers to be clustered?

Software information to consider during the scope phase

Equally important as the hardware information is the software stack that will be used during the performance assessment. You should ensure you document the following information:

- Operating system for each computer (32 or 64 bit, clustered or not).
- Server software to be installed on each computer.
- Virtualization software used.
- Specific software features not typically installed such as COM+ rollup fixes or SQL Server host fixes that are required.

Determine if code changes are within the scope of the performance assessment

This is one of the most important things to determine during the scoping process. BizTalk Server and the underlying components it uses (SQL Server, Windows, IIS and many more) can be optimized using the tuning techniques and configuration changes discussed in this guide. However, if an application has not been developed for optimal performance, it can hamper these performance gains. Therefore, code changes should only be removed from the scope if you have confidence a thorough code review has been completed before the application enters the lab. If

necessary, you may agree that only certain changes be allowed, for example, the removal of persistence points within orchestrations.

Roles and responsibilities

One of the most important areas of running a performance assessment is to ensure the roles and responsibilities are clearly assigned. The following key roles should be assigned at the onset of the performance assessment:

Engagement lead - The engagement lead is responsible for owning the overall engagement end-to-end. This role will typically be performed by a Senior Consultant or Architect. Ideally this person should have experience in tuning BizTalk Server based systems. Knowledge of SQL Server and any other technologies including third party ones is desirable. Please note it is not necessary that the lead is an expert in all areas, but they need to have at least a working knowledge of the technology in order that they can work with the specialists in those areas and understand the optimizations that they are applying.

Test designer - It is necessary to have someone who is dedicated to designing and implementing the tests that will be executed during the lab. This will involve deciding on the test framework that will be used to create tests, testing each of the tests to ensure that it will fulfill the requirements of the lab and ensuring that those responsible for running the tests are aware of how to use the client.

Environment owner - Having a representative environment within the lab that accurately reflects the production BizTalk Server environment is critical. The person performing this role will be responsible for checking each item of the software and hardware stack used and validating that there are no significant differences. For example SQL Server 2008 R2 when running on the 32-bit platform can only address 4GB of physical memory without using either Physical Address Extensions (PAE) or Address Windowing Extensions (AWE). In SQL Server 2008 R2, 64-bit up to 1 terabyte of memory can be addressed (this is the current maximum physical memory supported by Windows Server 2008 R2). Therefore a significant difference between the customer and the lab environment would be the architecture of CPU used by BizTalk Server and SQL Server. In addition to these obvious factors, other less obvious factors, such as service pack levels and hotfixes installed, can impact performance of the environment.

Build environment lead - After a detailed specification has been drawn up for the performance assessment, someone should be assigned the responsibility of building the environment. This will include the hardware and software stack. In many cases one individual will be responsible for this (this is often the environment owner). However in a large enterprise the environment owner may need to be the liaison between different teams, who may each be responsible for various components of the solution. For example, one team may be responsible for the Windows build, another for SQL Server, and yet another team for BizTalk Server.

Deployment lead - It is necessary to have someone responsible for ensuring that the application is deployed correctly to BizTalk Server, that the correct bindings are configured and that any custom configuration is applied. This role will require a thorough knowledge of the solution

and will require the knowledge to package an application and deploy an application and then validate that it is in a functioning state within the lab environment.

Product specialists – It is important to identify what product specialists are required for the performance assessment. The exact skills required depend upon the design and purpose of the BizTalk Server application.

One of the most common product specialist skills required is extensive knowledge of performance tuning SQL Server. These skills are required for two purposes: first, it is often necessary to perform SQL Server optimizations to optimize performance of the BizTalk databases and second, many BizTalk solutions make use of custom databases. The use of custom databases can become a bottleneck in the solution if the correct optimizations are not applied to the SQL Server environment. In order to identify and apply the appropriate database optimizations, the following SQL Server skills are typically required:

- Thorough understanding of SQL Server stored procedure code, including the ability to optimize existing stored procedures.
- Ability to identify and build missing database indexes.
- Ability to write scripts to split databases into multiple files.

  Note

  For more information on applying this optimization, see Optimizing Filegroups for the Databases.

- Ability to identify performance problems using SQL Server 2008 R2 Performance Dashboard Reports.

  Note

  SQL Server 2008 R2 Performance Dashboard Reports is available for download at http://go.microsoft.com/fwlink/?LinkId=118673.

For each specialist technology involved in the performance assessment, a list of requirements should be defined as is done above for SQL Server. This ensures the resource obtained has clear expectations of what will be required of them. Another technology that frequently requires specialized knowledge during the performance assessment is IBM Websphere MQ. The list below illustrates the requirements specification for an IBM WebSphere MQ product specialist:

- Experience in the monitoring, maintenance, and customization of MQSeries.
- Experience with installation and migration of new versions of MQSeries.
- Ability to analyze and tune MQSeries performance.
- Perform MQSeries problem analysis.
- Knowledge of the processes and procedures related to MQSeries security, administration, recovery and automation.

Documentation lead - Continuously updating the lab documentation end-to-end throughout the performance assessment is vitally important. The overall successfulness of a lab engagement should not be judged until the optimizations have successfully been applied in

the production environment and the system has obtained the desired level of performance. In order to do this, it is essential that a detailed record of the following information is kept:

- High-level results summary of the lab
- Unresolved issues
- Resolved issues
- Timeline for the lab
- Lab progress
- Implemented optimizations by category
- Implemented optimizations in chronological order (to ensure that they can be applied in the same order in the production system)
- High-level architectural diagram
- Detail of the scenarios to be tested
- Third party technologies involved
- Lab hardware diagram
- Contact list
- Detailed hardware inventory
- Appendix with full detailed results

Developer - Whether a developer is required depends very much on the scope of the engagement. If the code base has been locked down and the lab is there to test infrastructure and platform optimizations only then the services of a developer should not be required. This type of scenario can occur when performance testing is performed just before the “go-live” date of the production server. By this time, the code should have been locked down and full regression testing should be complete or in progress. Any changes to the code introduced in the lab could introduce a regression and, therefore, introduce risk to the production system. If code changes are permitted, then typically a developer will be required. The list below represents the skills typically required by a developer that is engaged in a BizTalk Server performance assessment:

- Ability to identify and fix performance issues with orchestrations
- Ability to identify and fix performance issues with pipelines
- Familiarity with .NET including:
    - Enterprise library
    - Visual Studio F1 profiler expertise
    - Microsoft Visual Studio 2010 Ultimate or Visual Studio Test Professional 2010

Test lead - Ensuring an accurate, complete and thorough set of results is obtained is critical. The test lead is responsible for ensuring the required information is captured, analyzed appropriately, and distributed appropriately after every test run. It is important to consider how to capture the data, typically the test data is suitable for presentation using an Excel spreadsheet. The following list illustrates the data that should be captured for each test that is run during the lab:

- Test run number

- Date
- Total messages processed
- Messages processed per second
- Time started
- Time stopped
- Test duration in minutes
- Suspended messages / Total messages processed – This can be captured either from the BizTalk Administration console or by measuring the BizTalk Server performance counters using Performance Monitor.
- # of messages that have failed processing
- # or message that have been successfully processed
- Test client responses
- Client process duration average, measured in seconds - Typically this value is measured when implementing a synchronous messaging solution with BizTalk Server, in this case it is important to know the value for average client duration as this is typically representative of how long an end user is waiting for a response from the solution.
- Client process duration maximum value, measured in seconds
- Client process duration minimum value, measured in seconds
- BizTalk Server request/response latency, measured in seconds
- Orchestrations completed per second
- % of messages processed on time
- Value of **TestResultsLocation** variable used by Visual Studio Team System testing tools.
- Value of **TestResultsLocation** variable used by BizUnit.
- Any comments or recommendations

As well as collecting the results, the test lead should ensure that they monitor each test run to see if there are any trends. Performance improvements should be communicated to the rest of the team and should indicate how much performance was improved and which optimization was applied to achieve the improvement. At the end of day it is important that the test lead provides a summary of the tests that have been performed in the lab. This allows the stakeholders to be kept informed of the continued progress of the lab. The table below illustrates how this information might be put together in a sample update e-mail:

Test results summary

Status	Throughput	Average Latency	%< 2 seconds	# of Test runs	# of BizTalk Server Computers	# of Messages	Average Message Size	Duration
Scale out	140 messages/second	0.777 seconds	99.3%	2	6	270,000	609 bytes	30 minutes
Best	50 messages/second	1.12 seconds	99.12 %	17	2	360,000	609 bytes	2 hours
Baseline	30 messages/second	1.52 seconds	92.9 %	4	2	36,000	609 bytes	20 minutes
Goals	5 messages/second	< 2 seconds	90%	-	2	-	-	-

Define all deliverables that are required at the onset of the performance assessment

It is important to agree upon deliverables that must be in place before embarking on a BizTalk Server performance assessment. The section below describes the deliverables that should be in place at the onset of the performance assessment.

Application to be tested – The complete application that will be used during the performance assessment must be available. It is important that the application is in a fully functioning state before beginning the performance assessment, otherwise there is risk of losing valuable lab testing time.

Planning for automated build and deployment - Before engaging in the performance assessment, an automated build and deployment process should be developed for the BizTalk Server solution to be tested. Automating the build process for an application enables you to do a continuous daily build process, which can then be accompanied by a set of build validation tests (BVTs) which test the functional flows through the system. This enables you to detect compilation and functional problems quicker and easier throughout the development lifecycle. Within the lab, this means that if any code changes are required you can quickly verify that the updated solution works without problems. Manually building, deploying and configuring an application for a performance lab can be a tedious and error prone task. Therefore it is recommended that an automation technique be used to accomplish the following build and deployment activities:

- Get latest code from source control.

- Build the complete solution.
- Deploy the solution to the environment.
- Create BizTalk applications.
- Add BizTalk Server resources such as .dll files and bindings to BizTalk applications.
- Import binding file to create ports and bind to orchestrations.
- Export BizTalk applications as a Microsoft® Windows® Installer package so that it can be deployed in the testing or production environment.

Note

For more information about implementing an automated build process, see Automating the Build Process

MSBuild was introduced with the .NET framework 2.0 to enable developers to automate tasks such as those described earlier. Several BizTalk Server–specific MSBuild tasks are included with the SDC Tasks library, which is available for download from http://go.microsoft.com/fwlink/?LinkId=119288.

Test data to be used during the performance assessment – The test data that is used has a considerable influence on the overall effectiveness and success of a performance assessment. Consider the scenario where a BizTalk Server application utilizes messaging, orchestration and the Rules Engine. The Rules Engine is called from within a pipeline component on the receive side to determine which orchestration will be used to process the message; and it is also called from within the orchestration at various points to determine flow. The Rules Engine implements caching so that rules policy execution is optimized. Therefore, if a single message is used as test data during the performance assessment, test results may be skewed (due to caching) and you may obtain results that can't be replicated in production.

Ideally the test data used during the performance assessment should be actual production data or a subset of production data. The test data should also simulate the load and pattern of traffic that will be flowing through the production system. Consider the following factors when defining requirements for test data:

- **Number of messages that will be flowing through the system in a given period** - This is normally expressed as messages per second or messages per hour.
- **Types of messages** – Are the messages flat file, XML, or binary?
- **Distribution of messages** – What percentage will be flat file, what percentage of each XML message type will be used?
- **Peak load processing requirements** - In many scenarios a large interchange may be processed during non-peak hours. For example a large batch of payments may be posted to a bank's systems at midnight. If this is the case, ensure you are able to replicate this during testing.
- **Endpoints used to receive/send messages** - In many environments separate receive locations are configured to receive different types of messages. For example flat file messages may be received using the File adapter or the MQSeries adapter may be used to receive XML messages. While messages may all be processed by the same

orchestration(s) they will have different entry points into the system. Each of these different receive locations can be hosted in a separate host; in fact doing this will often improve the overall performance of the system.

The table below provides an example of the information that should be captured when determining test data specifications:

Sample test data specification

Messages per second	• Maximum throughput is key in this scenario • Goal of 50 messages per second • Low-latency is not a goal
Type of messages	• Small XML messages, approximately 25 KB that conform to XML schema *X* • Medium XML messages, approximately 512 KB that conform to XML schema *Y*
Distribution of messages	• 58% 25 KB (small XML messages) • 25% 512 KB (medium XML messages) • 11% 3 MB (medium binary data – PDF) – non compressible • 4% 7.5 MB (large binary data – PDF) – non compressible) • 2% 20 MB (large binary data – PDF) – non compressible
Peak load processing requirements	Large binary data (20MB) will represent approximately 2% of data (as specified above) the time at which this is processed is non-predictable. The system must be able to accommodate these large messages at any given time.
Endpoints used to receive/send messages	**Small XML messages (25 KB)** • Receive location: PaymentXMLDocIn • Host: ReceiveHost • Pipeline used: XMLReceive **Medium XML messages (512 KB)** • Receive location: PaymentXMLDocIn • Host: ReceiveHost • Pipeline used: XMLReceive **Medium binary data (3 MB) – PDF – non compressible**

	• Receive location: BinaryDataIn • Host: ReceiveHost • Pipeline used: PassThruReceive **Large binary data (7.5 MB – 20 MB) – PDF – non compressible** • Receive location: BinaryDataIn • Host: ReceiveHost • Pipeline used: PassThruReceive
Location of test data	Locally accessible test data file share, for example: \\PerformanceLabs\July\Test Data\

Putting the information into a table accomplishes several things. First off, it makes it easier for stakeholders to agree on assumptions made about the test data. Second, it provides you with information that can be used to decide on potential optimizations for the performance assessment. For example in the table above you can see that all the receive locations used to process all different data types are hosted within the ReceiveHost BizTalk host. This means that each instance of this host will be responsible for processing different types and sizes of data (e.g. XML and binary non-compressible PDF data). Given that each host instance is a single instance of the BizTalk Server process (BTSNTSVC.EXE), this could become a processing bottleneck. Therefore in this scenario one immediately obvious optimization for the environment would be to test the performance improvement of separating each receive location into its own individual host. Having access to the test data information in a summary tabular format makes it easier to gauge simple optimizations such as this.

Planning for automated load tests and load generation - After the test data profile for the performance assessment is established, it is important to consider how to perform load testing within the environment. For BizTalk Server 2010 load test, we used Visual Studio 2010 load test. For more information about how to facilitate load testing using Visual Studio 2010, see Using Visual Studio to Facilitate Automated Testing.

See Also

Phases of a Performance Assessment

Phase 2: Planning the Assessment

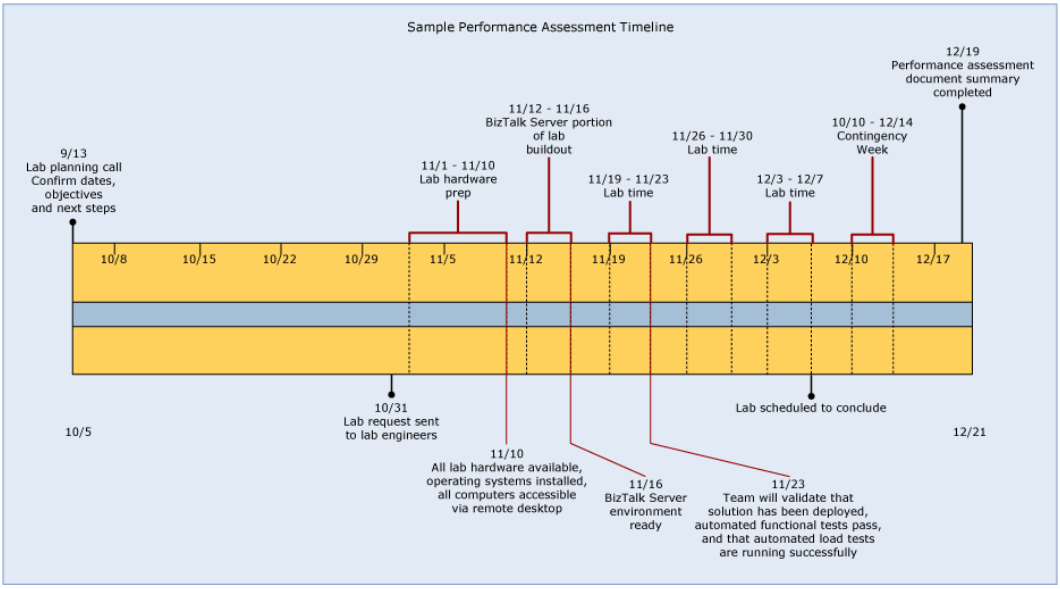
The Plan phase of a performance assessment largely consists of setting up specific milestones for the completion of the deliverables identified during the Scope phase. The Plan phase is the “when” to the Scope phase’s “what.” Third-party software and physical logistics are “how” and “where” aspects that should also be considered during the planning phase as these aspects may require additional lead time to implement.

This topic describes the various aspects of the Plan phase of a performance assessment.

Solution milestones timeline

It is important to have a clear and concise record of the milestones for the performance assessment. Setting specific milestones will increase the likelihood that the solution will be completed in a timely manner. Consider creating a Visio timeline to visually represent dates, milestones, and tasks associated with the performance assessment. Keep the timeline simple enough to be easily understood yet detailed enough to communicate milestones effectively. The actual milestones should be agreed upon by all stakeholders and reviewed regularly. A sample Visio timeline is displayed below:

Sample performance assessment Visio timeline



Note

For more information about how to create a timeline using Visio 2010, see the article Project timeline (http://go.microsoft.com/fwlink/?LinkID=208367).

The following table summarizes the deliverables and other details associated with the milestones in the preceding figure.

Milestone	Deliverables	Other Details
Lab planning call	Begin documentation	Confirm objectives defined during scoping, assign dates to deliverables to create task and milestones.
Lab request	Hardware/software diagram and specification	The hardware/software diagram and specification is required by the lab team to procure and

Milestone	Deliverables	Other Details
		build the hardware to the designated software specification.
Lab hardware available	Lab hardware should now be accessible to all stakeholders.	For illustration purposes, it has been assumed that a system build team is available to build out the lab. In reality this may not be the case and responsibility for building out the lab hardware may rest upon one or more of the members of the team participating in the performance assessment.
BizTalk Server environment ready	• BizTalk Server should be fully deployed and configured. • The BizTalk Server application to be tested should be fully deployed.	
Functional validation of solution completed	All functional flows through the system should be tested at this point.	Ideally this will be done using automated functional testing so that this process can be easily repeated as required.
Lab scheduled to conclude	• Performance goals should be met at this point • Documentation should be up to date with all optimization and configuration settings that have been applied so far.	
Performance assessment document summary completed	Final lab report is delivered	

Non-Microsoft software considerations

It's a good practice to be very clear from the beginning about which stakeholders will provide the needed expertise for any third-party software that will be used during the performance assessment. Consider the following when non-Microsoft software will be used with the solution:

- Determine how the software or hardware required be obtained.

- Plan for capacity and sizing to ensure that non-Microsoft software does not become a bottleneck in your solution.
- Determine a plan of action for installing required non-Microsoft software.
- Create a plan of action for configuring and optimizing required non-Microsoft software.

Physical space and other logistics

Physical space is an important detail to consider during the planning phase. In order to conduct a lab, computers need to exist where they can be accessed and the participants will need space to work and hold discussions. The participants will also benefit from additional space where phone conferences may be held in a less public forum.

One of the most important aspects of the physical environment for the lab is to create an environment where all participants can comfortably see the same set of screens. This can be accomplished by using overhead projectors through which the team can view remote desktop sessions used during the lab or by shadowing sessions using the “shadow” feature of remote desktop.

Performance assessment engagements may yield unexpected problems, which will only be compounded by the fact that the engagement is typically on a tight schedule. Because of the sometimes lengthy working days that are required to meet milestones, physical space logistics will become important to the success of the engagement.

In summary, the following factors should be considered when planning the physical space required for the performance assessment:

- Lab computers must be accessible to team members.
- Team members will need space to hold discussions during the lab.
- Team members should have access to visual media that allows all members to see the same set of screens, typically either an overhead projector tied to a remote desktop screen or by using the “shadow” feature of remote desktop.
- Team members should have access to a “whiteboard area” which will be used to facilitate discussions during the lab.
- Team members should have access to space where phone conferences can be held in a less public forum.
- If possible, the lab area should be climate controlled during extended hours, this will accommodate off hours access if required.
- Everyone will likely still need to communicate with their day jobs, so allow enough table space for both client computers and personal laptops, 2 for each participant.
- Ensure that enough power strips and network connections are available in the lab. If wireless is used, arrange guest access for participants.
- Ensure that the lab environment will be secure during off hours.

See Also

Phases of a Performance Assessment

Phase 3: Preparing for the Assessment

The Prepare phase of a performance assessment can be thought of as the “how” to the Scope phase’s “what” and the Plan phase’s “when.” At this point in the performance assessment, all stakeholders should have agreed upon the scope of the engagement and the plans for conducting the lab. It is in the Prepare phase of the performance assessment where the plans are executed and steps are taken to get ready for execution of the performance lab.

This topic describes the various aspects of the Prepare phase of a BizTalk Server performance assessment.

Detailed design of the solution platform

A detailed solution design facilitates communications and avoids assumptions, which will improve the agility and effectiveness of all activities. You should fully document the following elements:

- **BizTalk Server databases and how they will be distributed across computers** - SQL Server performance is one of the key factors in overall BizTalk Server performance. If SQL Server is experiencing resource constraints, this will impact the ability of BizTalk Server to process messages. The main factor that influences BizTalk database performance is the speed of disks that they are hosted on. Separating the transaction log and database files for each BizTalk database onto separate drives or SAN LUN’s has been shown to remarkably improve the overall performance of BizTalk Server. Therefore, it is important that this information be recorded in an easily accessible manner. The values that will be used in the production environment should be documented in the detailed solution design. The following table provides an example of how this can be done.

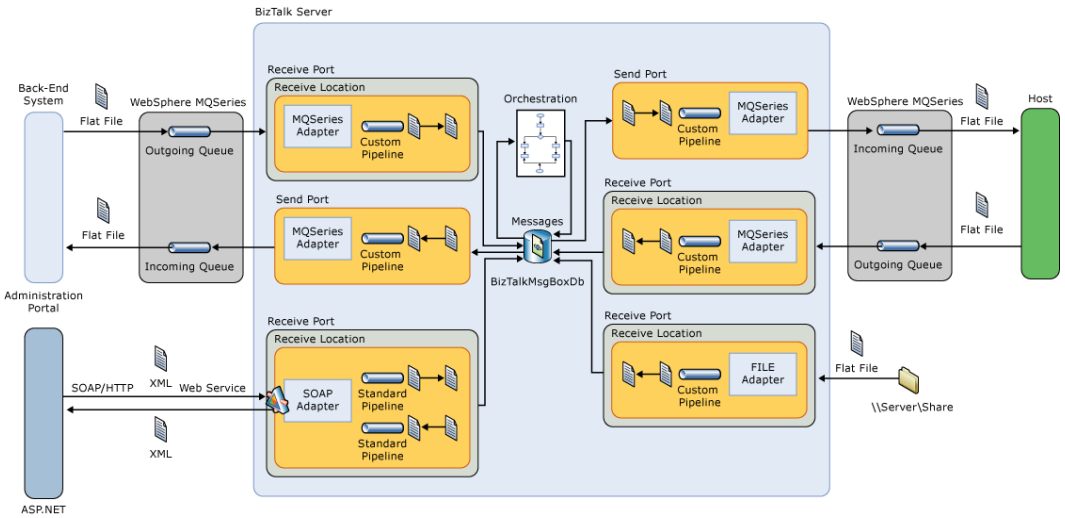
BizTalk Database	Volume Name	Files	LUN# or ML_#	Physical LUN Size (GB)
MessageBox	Data_TempDb_1	TEMPDB, MASTER, and MSDB data files	1	134
	Logs_TempDb_1	TEMPDB, MASTER, and MSDB transaction log files	2	134
	Data_BtsMsgBox	BizTalkMsgBoxDb data file	3	134
	Logs_BtsMsgBox	BizTalkMsgBoxDb transaction log file	4	134
BAM	Data_TempDb_2	TEMPDB, MASTER, and MSDB data files	5	67
	Logs_TempDb_2	TEMPDB, MASTER, and MSDB transaction log files	6	67
	Data_BAM	BAMPrimaryImport data file	7	134
	Logs_BAM	BAMPrimaryImport transaction log file	8	134
BizTalk Tracking, Management, Single Sign-On, and Rule Engine databases	Data_TempDb_3	TEMPDB, MASTER, MSDB, BizTalkDTADb, BizTalkMgmtDb, ENTSSO, and BizTalkRuleEngineDb data files	9	67
	Logs_TempDb_3	TEMPDB, MASTER, MSDB, BizTalkDTADb, BizTalkMgmtDb, ENTSSO, and BizTalkRuleEngineDb transaction log files	10	67

- **BizTalk Host design and descriptions of each host and their instances.**
- **Description of each orchestration.**
- **Description of each pipeline.**
- **Description of custom components such as .NET assemblies and COM+ components.**

Detailed architecture diagram

The following diagram illustrates an architecture diagram that could be used for a performance assessment.

BizTalk Architecture Diagram



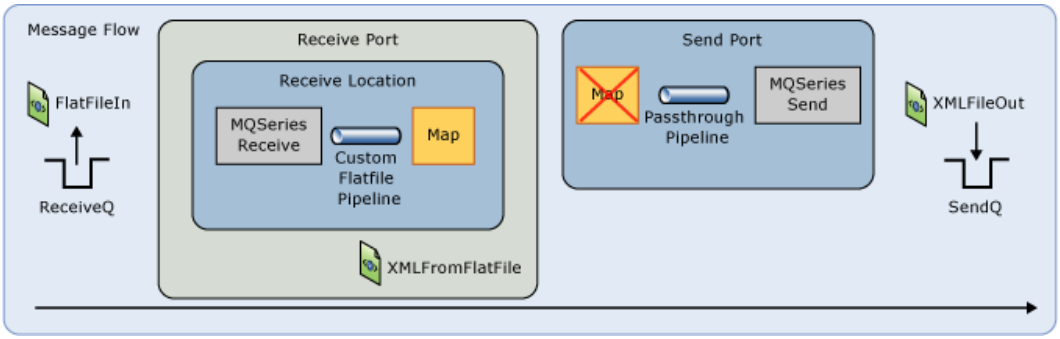
Message flow diagrams

Create detailed message flow diagrams to help prevent confusion or false assumptions regarding what is supposed to be happening to messages during processing.

When thinking about a BizTalk solution holistically, we tend to think of the message flow through the system. This Message Flow perspective is especially important when doing performance testing because all parts of the flow must be considered as potential bottlenecks. Having a message flow diagram prevents any confusion or false assumptions regarding what is supposed to be happening to messages during each test run.

In the following example, created using simple Visio shapes, everyone on the project regardless of background can quickly understand how a message gets into the system, what parts of the solutions interact with the message, and finally where the message lands after processing.

Message Flow Diagram



The following details should be considered when creating the message flow diagrams:

- Describe the lifecycle of each type of message from the time it arrives at a receive location until all resulting messages are sent and all related processing is completed.
- Describe how processing changes for error conditions.
- Include details about correlation, delivery notifications, and acknowledgements.
- Include details about dependence on external systems.
- Include performance requirement information regarding latency and throughput.

Third-party software details

All non-Microsoft software that is used should be fully documented as part of the detailed solution design.

Detailed lab hardware stack

Building on the previously created high-level hardware diagram, the following hardware information should be fully documented:

- Processors
  - Type
  - Speed
  - Number of cores
  - Hyperthreading
- Memory
  - Amount
  - Speed
  - Parity
- Network
  - Number of network interface cards (NICs)
  - Speed of network
- SAN
  - Number of SAN cards in each computer
  - Number of logical unit numbers (LUNs) for each computer and purpose for each LUN
  - Speed of storage area network (SAN) Cards
  - SAN card configuration details
  - SAN disk allocation, formatting, and partitioning
- Disk
  - Local disk details for each computer
  - Formatting used for local disks
  - Partitioning details for local disks
- Cache
  - L2 Cache amount

- L3 Cache amount

Detailed lab software stack

The following software information should be documented:

- Specific operating system versions, editions, and architecture
- Specific operating system features
- Specific software installed on each computer
- Specific drivers
- Service Packs and other updates
- Configuration values for all software and operating system features used if they vary from default values

See Also

Phases of a Performance Assessment

Phase 4: Building the Assessment Environment

The Build lab phase of a performance assessment is used to install the hardware and software for the environment in conformance to the design decisions made in previous phases. Because the Build lab phase can be time consuming, it is not unusual for this phase to overlap earlier phases. In many scenarios, the hardware and operating system may be installed before a final decision has been made as to the application architecture. The Build lab phase of a performance assessment typically includes the tasks discussed in this topic.

Obtain all build-lab infrastructure at least a week in advance of the lab start date

Plan to have available all of the required hardware and software at least a week before the lab start date. This will ensure that valuable lab time is not wasted for want of the required infrastructure.

Configure third-party software systems

There may be third-party systems that need to be built out and configured before the lab can begin. If subject matter experts (SMEs) are required for these systems, be sure they are scheduled during the build-out and lab execution stages. Ensure that they thoroughly document their build-out procedures.

Install and configure the BizTalk Server environment

Detailed instructions for installing BizTalk Server and the required dependency software can be found in the BizTalk Server 2010 Installation and Upgrade Guides (http://go.microsoft.com/fwlink/?LinkID=191321). After successfully installing and configuring the BizTalk Server environment, complete the following tasks:

- Follow the recommendations listed in the Operational Readiness Checklists (http://go.microsoft.com/fwlink/?LinkId=160134).
- Follow the recommendations in Optimizing Performance.

- Ensure all computer times are properly synchronized.
- Verify MSDTC functionality between all computers in the environment.
- Ensure any custom tracing/logging is disabled unless absolutely needed.
- Install the Visual Studio 2010 Ultimate edition for load testing. For more information about how to perform automated testing using Visual Studio, see Using Visual Studio to Facilitate Automated Testing.
- Setup Performance Monitor counters and logs, as needed.
- Setup a debugging computer to debug the solution if code changes are within the scope of the performance assessment.
- Defragment all hard drives.
- Disable antivirus real time scanning.
- Back up the Enterprise Single Sign-On Master Secret.

Install the BizTalk Server application to be tested

Installation of the application to be tested will typically include the following steps:

1. Use the BizTalk Server Administration console to do the following:
   - Create Hosts
   - Create Send/Receive Handlers.
   - Create Host instances.
   - Create BizTalk Server Applications.
2. Application installation:
   a. Deploy BizTalk Server binaries to the BizTalk Server group.
   b. Import bindings to the BizTalk Server group.
   c. GAC BizTalk Server and non-BizTalk Server binaries on all boxes.
   d. Ensure dependency components exist on all boxes.
3. Install dependency applications.
4. Configure transports and physical endpoints in the BizTalk Server Administration console.
5. Start services.
6. Perform basic smoke tests – Smoke tests are end-to-end functional tests that test the basic functionality of your solution.

Implement automated build and load testing

Implementation of an automated build and load testing process is arguably the cornerstone of a BizTalk Server performance assessment. An automated build process should be implemented if code changes are within the scope of the performance assessment. Automated load testing should be implemented for all load testing scenarios. The initial time investment required to implement automated build and load testing is quickly recouped, automation accommodates rapid and precise repetition of mundane build/testing tasks that are subject to human error. For more information about implementing an automated build and testing process, see Implementing Automated Testing in this guide.

Configure performance monitoring

Accurate performance monitoring is critical to the success of the performance assessment. Determine which performance metrics should be evaluated based on the throughput and latency goals that were defined in the Scope phase. Performance monitoring should be performed on each computer in the BizTalk Server environment. For more information about the performance counters available for BizTalk Server 2010, see Performance Counters (http://go.microsoft.com/fwlink/?LinkID=157269) in the BizTalk Server 2010 Help. Use the Performance Analysis of Logs tool (PAL) to generate HTML reports that graphically chart important performance counters and generate alerts when thresholds for these counters are exceeded. For more information about the Performance Analysis of Logs (PAL) tool, see Performance Analysis of Logs (PAL) tool (http://go.microsoft.com/fwlink/?LinkID=98098).

Establish and document the solution's baseline performance

Baseline performance should be calculated so that the effect of performance optimizations applied during the performance assessment can be measured.

See Also

Phases of a Performance Assessment

Phase 5: Executing the Assessment

The Execute phase is where the performance data is generated through automated load testing and where performance bottlenecks are discovered and addressed. This phase has an iterative process whereby performance bottlenecks are reduced or eliminated by testing, evaluating performance, optimizing, and testing again.

This topic describes the steps that are typically followed during the Execute phase of a BizTalk Server performance assessment:

Step 1: Run automated tests

Begin the Execute phase by running automated tests. A BizTalk Server performance assessment typically focuses on throughput and/or latency testing but may include build verification and functional tests. For comprehensive information about running automated testing, see Implementing Automated Testing.

Step 2: Document and evaluate test results

At the conclusion of each test, thoroughly document the test results and evaluate whether the stated performance goals have been met.

Step 3: Modify the configuration of BizTalk Server, third-party systems, or solution artifacts to tune for performance based on the test results

Use the test results to make decisions about how to optimize the environment to reach stated throughput or latency goals.

Step 4: Record all changes made to the environment

Ensure that any changes made to the environment are thoroughly documented. A record of the changes will allow you to easily apply the changes in the production environment and will accommodate the undoing of changes if need be.

Step 5: Repeat this cycle until goals are achieved

Continue the cycle of testing, evaluating and documenting results, optimizing the environment, and documenting changes to the environment until the desired performance goals are reached.

Step 6: Summarize test results at the end of each day

Complete an “executive summary” of the test results and progress made at the end of each day. Compile an e-mail that contains the summary, and send to all stakeholders in the performance assessment to keep everyone informed of progress that is being made.

Step 7: Update the project plan as timeline goals are met

The timeline developed in the Plan phase is a good reference for the overall progress of the performance assessment. Update this timeline as necessary to communicate the progress to all stakeholders.

See Also

Phases of a Performance Assessment

Implementing Automated Testing

BizTalk Server solutions are often deployed in “mission-critical” scenarios, whereby the BizTalk solution is a core component of the business. In these scenarios, the continual performance and stability of the BizTalk solution is a key business requirement; if the BizTalk solution fails, the associated downtime costs are significant. In such scenarios, it is of paramount importance that the BizTalk solution be thoroughly tested before the solution is placed into production. The costs associated with properly testing the solution are small compared to the downtime costs resulting from not testing or insufficiently testing the solution. Therefore, the testing of a BizTalk Server solution is arguably the most important phase of any BizTalk Server solution deployment.

This section describes the proper methodology for testing a BizTalk solution before running the solution in a production environment.

In This Section

- Why It Is Important to Test
- Automating the Build Process
- Using BizUnit to Facilitate Automated Testing
- Using Visual Studio to Facilitate Automated Testing

Why It Is Important to Test

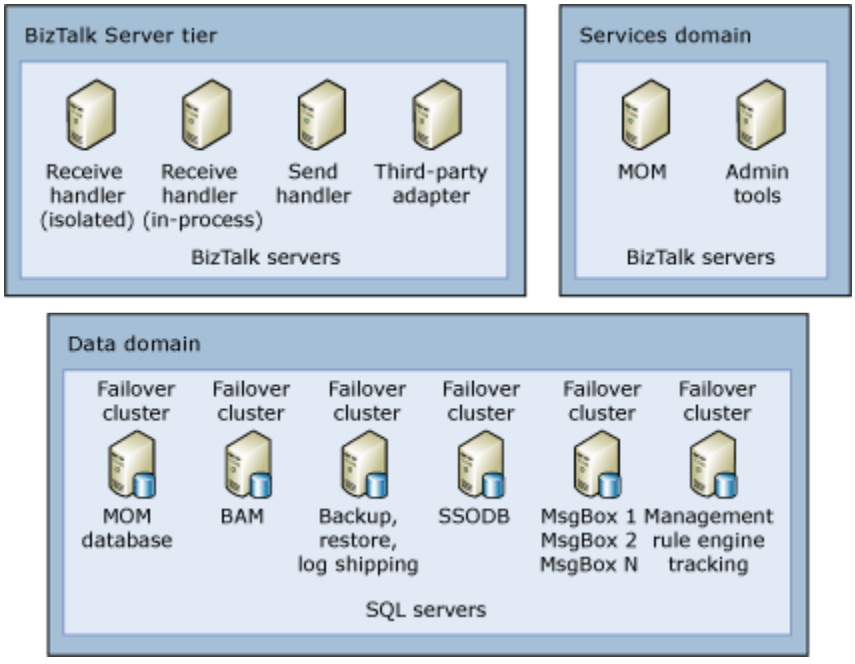
This topic provides an overview of the mindset that leads to insufficient testing, describes the risks associated with failing to test BizTalk solutions, and contrasts the pitfalls of manual testing with the benefits of automated testing.

Testing as "overhead"

Unfortunately, with ever increasing demands for return on investment (ROI), the testing phase of a project is often seen as one of the first aspects of a project plan that can be scaled back.

One argument against testing BizTalk solutions is that "We don't need to test our solution because Microsoft already thoroughly tests BizTalk Server." While it is true that Microsoft does thoroughly test BizTalk Server, different usage scenarios require one of an almost countless number of permutations of business requirements for throughput, high availability, adapter usage, latency, tracking requirements, orchestration usage, and custom code. Because BizTalk Server is extremely flexible and can accommodate so many different usage scenarios, it is simply not possible to anticipate and test all of them. Furthermore, the default settings that are applied in a BizTalk Server environment should be optimized to accommodate each usage scenario. The only way to determine the optimal settings for a particular usage scenario is to test the solution, measure various parameters, tune the environment, and retest. Consider the following diagram, which depicts a sample physical architecture for a BizTalk Server solution.

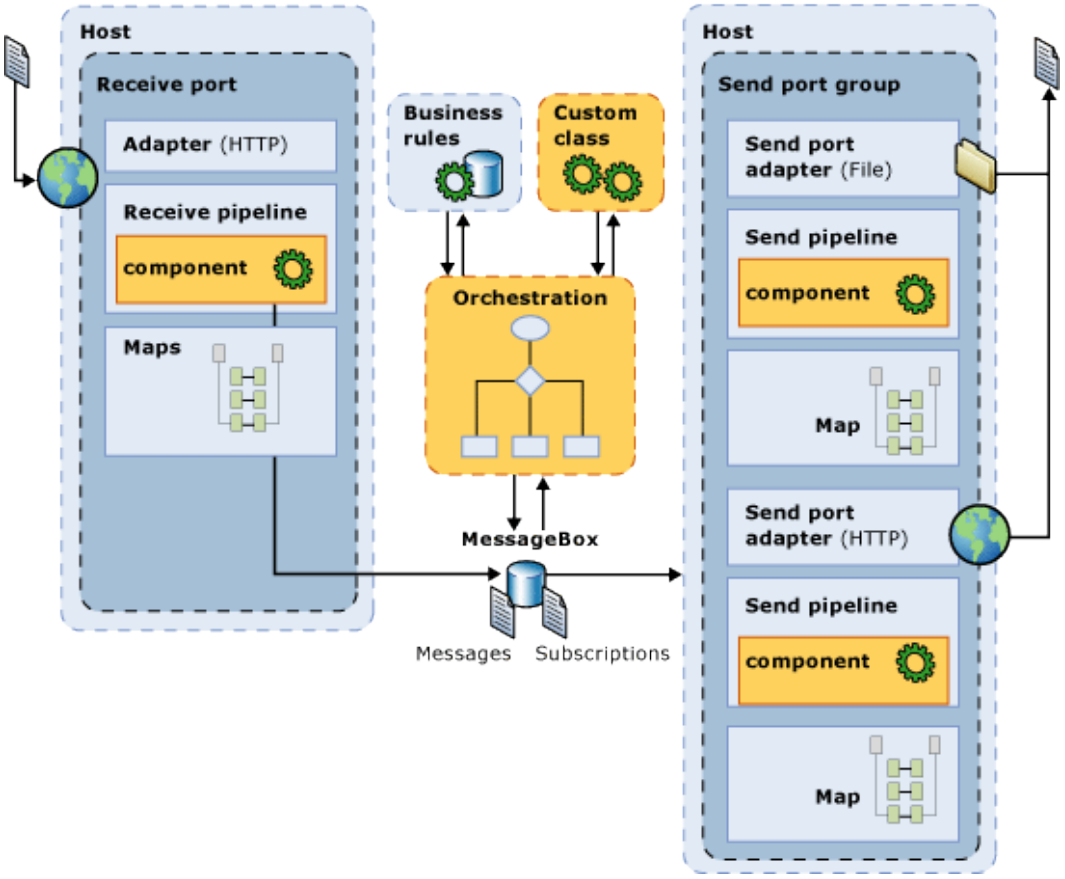
Sample Physical BizTalk Architecture



You can see in the diagram above that the BizTalk Server solution contains many moving parts, including multiple computers running BizTalk Server, computers running SQL Server, server cluster nodes, and Active Directory domains. After the system is up and running, there will be many configuration tweaks applied to optimize the application per the business requirements, as

to maximize the overall ROI of the solution. This single scenario shows that there are many variables in play. In addition to the physical architecture of the environment, consider the following diagram, which depicts the flow of a message through BizTalk Server.

Sample Logical BizTalk Message Flow



When we look at the logical diagram of a message flowing through BizTalk Server, we see other variables that necessitate per-project testing, including custom send and receive pipeline components and custom classes that can be called from BizTalk orchestrations. Given that the type complexity and use of custom components and BizTalk components varies from project to project, it becomes more evident why it is important to perform testing for each specific usage scenario.

Testing methodology and timelines

To ensure testing is performed effectively and efficiently, the test harness should be fully automated so it is easily reproducible and minimizes the chance for human error. Additionally, adequate time should be allotted for testing when planning the project. A minimalist approach to testing would comprise manual steps similar to the following:

1. Manually load one or more messages into a receive end point, such as a file drop or by using a SOAP client to call a Web service.

2. Manually validate the correct content and structure of a message. Because multiple schemas are often present in a project, the messages may be a mixture of flat file and XML and may also contain cross field dependencies.

   Note
   An example of this would be any project involving SWIFT messages. These are flat file messages that have cross field dependencies. That is, one field's value depends on another – simple XSD validation will not do here; hence the SWIFT Accelerator makes use of the BizTalk Rules Engine for validation of the messages. For more information about the SWIFT Accelerator, see SWIFT Accelerator (http://go.microsoft.com/fwlink/?LinkID=79657).

3. Manually check the event logs on the BizTalk Server computers for errors.
4. Manually check the BAM databases (if used) to validate that activity information is recorded correctly.

Using a manual process as described above for testing is subjective and error prone. Consider having to examine a hundred line SWIFT message with cross field dependencies for 10 different test cases. Most project developers would not be able to or even if they were able, would not be inclined to engage in such a task reliably and accurately. Implementation of a subjective, manual, error prone testing process adds risk to a project and increases the chance of failure.

The risk of failure is often amplified by project planning timelines that do not incorporate sufficient time for testing. All too often, a project testing strategy hinges on a single manual test pass which is scheduled a week or so before the go-live date. Such limited testing places the project at risk. Given such a limited timeline for testing, if any problems are detected, then the project may be delayed because no time has been allotted to fix problems. Furthermore, if a problem is discovered and fixed, there may be insufficient time left to perform subsequent test passes before the system goes live.

The reality of "single final build, single test pass, take the project live" testing is that it often results in the project being delayed, the project going over budget, or even worse, the project failing completely! For mission critical systems, this sort of testing methodology is a disaster waiting to happen.

Testing effectively and efficiently

Because appropriate testing is often viewed as an expensive luxury rather than a integral requirement, it is all too often tacked on at the tail end of a project. Given the complexity of the software and hardware stack used within heterogeneous enterprise environments, this approach is unrealistic. Implementation of an automated test approach that can be re-run on demand is the best way to test effectively and efficiently and is an integral component of successful projects that ultimately deliver an excellent ROI to the business. By investing at the beginning of the project in full end-to-end functional tests for each code path through the system you are generating test assets. These assets require investment (i.e. development time) in order to reap the benefits of increased testing effectiveness and efficiency later. To minimize the investment that is needed from the project to derive such a framework we recommend that you utilize the Visual Studio 2010 load test framework. Visual Studio 2010 includes the majority of the common testing steps

required for successful testing and is the framework used in the test scenarios presented later in this guide.

See Also

Implementing Automated Testing

Automating the Build Process

An automated build process compiles, deploys and then runs build verification tests (BVTs) against the latest source code for a project at regular, predetermined intervals. Then a "build report," which details the success or failure of the build process, is disseminated to the project stakeholders. The build report is analyzed to determine what areas of the project need attention and if the project should be rolled back to an earlier version/build.

The power of an automated build process is that it can be scheduled to be run during "off hours." This way, it can help ensure the stability of the project without taking cycles directly away from the development time.

This topic provides an overview of the build process, describes how build verification testing fits into the build process, describes aspects of functional testing used during build verification testing and provides information about the importance of automating the build process.

Overview of the build process

Before looking into the specifics of testing, it's important to consider the context of how testing fits into the overall build process. All of Microsoft's product groups operate from the philosophy that the build process is the heartbeat of any software project. This approach is also used by many of the world's top consulting firms that build mission-critical solutions on top of the Microsoft software stack. Without a steady heartbeat and a regular check of it, it is impossible to know the health of the project. To ensure that the product groups have a continuous insight into the overall health of the project, the build process is run daily, typically at midnight. The following steps summarize a typical automated build process:

1. Obtain the latest build of source code from the source code repository.
2. Compile the source code.
3. Pack up binaries for deployment – typically using scripts or Microsoft Windows Installer files.
4. Deploy project to a test server.
5. Automatically run a full set of build verification tests (BVTs).
6. Publish a detailed build report to members of the project team.

Note

BVTs are functional tests that exercise the main features of the solution to test its quality. You need to ensure your BVTs are comprehensive enough to gauge the build quality, yet small enough to execute within the allocated daily build time!

Note

You should also treat any deployment, un-deployment, configuration and installation scripts or processes as part of the software project for testing purposes. Both the

operations of the project, as well as the deployment and configuration of it, should be tested.

This process is typically completed by early morning around 6 A.M., which enables the first members of the team to start work on the new day's build. If one or more of the BVTs from the previous night failed, then it is the team's responsibility to fix it as soon as possible.

Following a daily build process has many advantages for the project. First, it provides a regular heartbeat (made up of the daily build plus the automated BVTs). Second, using BVTs forces integration with systems which is a tricky task, and doing this early in and of itself reduces project risks. Due to the time required to complete them, stress and performance testing are typically performed outside of the daily build process. Stress and performance tests are typically scheduled to be performed on a milestone build in the project.

The daily build process can be, and has been used, very effectively on BizTalk solutions. However, you need to ensure tasks that are typically left to the end in projects are done iteratively from the start. For example, deployment in BizTalk Server is certainly non-trivial. It is important that automated deployment scripts be developed up front. If you do not do this, you'll end up manually deploying and un-deploying the solution many times throughout the project, which will cost you more time in the end. There are tools available to drive the daily build process; Visual Studio Team System and Team Foundation Server are the primary choice for many people. MSBuild scripts may be used to drive the steps in the build process. Another alternative is the open source CruiseControl.NET tool (http://go.microsoft.com/fwlink/?LinkId=116093).

Note

Use of this tool is not supported by Microsoft, and Microsoft makes no guarantees about the suitability of this programs. Use of this program is entirely at your own risk.

For more information about automating testing using Microsoft Test manager, see the topic Running Automated Tests (http://go.microsoft.com/fwlink/?LinkID=208368) in the Visual Studio 2010 online documentation

For more information about automating the build process using Visual Studio 2010, see Building the Application (http://go.microsoft.com/fwlink/?LinkID=208369) in the Visual Studio 2010 documentation.

Build verification testing

Build verification testing usually comprises the following elements:

- **Unit Tests** - Because unit tests are typically the first tests to be developed, ideally they should be created before the code has actually been written, if you are truly using a test-driven development approach. By adding them into BVTs during the early stages of a project, you provide at least some code coverage immediately. As the number of functional tests and hence test coverage grows, you may opt to remove unit tests from the BVTs.
- **Smoke Tests** - End-to-end functional tests that test the basic functionality of your solution. If these fail, something is seriously wrong. These can usually be run relatively quickly.
- **Functional Tests** - These also target end-to-end scenarios, but in this case they test all the scenarios in the project. For large projects it may make sense to categorize your functional tests further (for instance, to enable a particular component to be tested quickly, reliably, and

in isolation). These functional tests should be locked down after you have signed off on your solution as being functionally correct.

- **Regression Verification Tests** - Every time a solution bug is found and fixed, regression verification test cases should be added to verify that the bug is fixed and that it is not reintroduced later in the project life cycle. These tests will typically be cases that were not covered in the functional test cases. You should expect that your regression verification tests will increase in number even after the solution has gone live, if new bugs are discovered and fixed.

On very large projects, you may need to make your BVTs a subset of the full functional test suite (due to length of time they take to execute). For smaller projects, BVTs will constitute the entire set. Obviously, if you are going to integrate the BVTs as part of your daily build, the whole process needs to be automated. In the rest of this topic, we will focus on how you can use BizUnit and other tools to accomplish this.

Functional testing

In the context of BizTalk applications functional tests, test a specific end-to-end scenario. When performing this type of testing, it is useful to imagine BizTalk Server as a black box that you test functionally from an external perspective. For example, a test may involve feeding an input message (with known values) to a receive location end point (for example, URL, FTP location, whatever your choice of transport is). The test would then monitor the correct number of messages with the correct output that is produced on the send side. This may sound relatively straightforward. However, when you consider that some scenarios require inputs to come in a certain order and at a certain time, and you compound this with other solution requirements (such as, when recording tracking data in BAM), it becomes clear that a tool and framework can be used to orchestrate this.

It is critical that functional testing is designed to cover all the possible paths through your solution. This should include not only those scenarios you expect in production, but also the failure paths and exception handling paths you have implemented but hope never to use – one phrase commonly used to describe this is testing for the “bad day scenario.” You should ensure all orchestrations, all permissible message types, and all code branches are exercised by your functional test suite. The following sections describe developing positive and negative functional test cases to cover all code paths.

For more information about functional testing and the other testing categories that should be implemented before placing a BizTalk Server solution into production, see the topic Checklist: Testing Operational Readiness in the BizTalk Server 2010 Operations Guide at http://go.microsoft.com/fwlink/?LinkId=160138.

Positive tests

- It is important when performing positive tests to ensure all combinations of messages, pipelines, orchestrations, and endpoints are passed through the solution so that all the message flows are exercised. To ensure that you test all code paths will likely require that you process different messages with different content.
- When testing, use the transport type that will be used in production. Unfortunately, all too often, functional testing is performed only by using the file adapter when some other transport

will be used in production. Adopting this approach is setting you and the overall project up for failure later on.

- Validate the payload of all messages that are sent out from the system. If the messages are XML, you should validate their schema and key fields in the message using XPath expressions.
- Validate any tracking data stored in BAM (if used); any other data that is left in external data repositories should be accounted for.
- Test the execution of all Business Rule Engine (BRE) policies and rules if your solution uses BRE.

Negative tests

- Ensure you test the handling of invalid messages through your system. You should verify that your chosen strategy (to reject them before they come into BizTalk Server, or to suspend them) has worked correctly.
- When testing the handling of invalid messages, ensure you test that any receive-side error handling orchestrations have been implemented to handle suspended messages.
- Ensure that your failure scenarios cover all exception blocks in your orchestrations. Failing to test this adequately is a common problem.
- If you are using long-running transactions with compensation behavior, test these scenarios very carefully. This is another area where inadequate testing will incur serious consequences in a production environment.
- Ensure failures are logged correctly in the appropriate error logs.

Automation is key

To do all of this efficiently and effectively, invest the time up front to automate testing. Manual testing is time consuming, error prone, and expensive (in terms of time and cost). Every time you perform a manual test pass, you add another batch of tasks that have to be handled by project resources (people in the team). By automating this up front, you are able to get a return on the initial investment that is required to develop the tests every time they are run. Good functional tests should execute quickly and efficiently and be repeatable; each test should also be autonomous (It should be independent of any other; adopting this approach enables you to run multiple tests sequentially as a test suite.) The functional tests should always produce the same result. Poorly written functional tests or poorly written code will result in different tests results between test runs, leading to confusion and wasted time investigating the root cause of the failure.

It is important to minimize the development effort required to write each functional test. Usually the more expensive it is to produce something (in terms of development time), the fewer test cases you are likely to end up with. This means you will have a lower level of test coverage over your code. By using a test framework, you can develop test cases quicker and easier and, hence, make it easier to get full code coverage. Most good test frameworks use a declarative approach to defining tests. (That is, the configuration for a test is stored in a configuration file, which is typically an XML file.) Using a good test framework enables you to develop a full functional test suite in an agile and reliable manner and avoids having to “reinvent the wheel” over and over, so to speak.

MSBUILD support for BizTalk Server projects

BizTalk Server 2010 provides support for the Microsoft Build Engine (MSBUILD) platform, which accommodates the building of managed projects in build lab environments where Visual Studio is not installed. MSBUILD accommodates building projects from a command line and advanced functionality including MSBUILD logging and batching. For more information about MSBUILD, see MSBuild Overview (http://go.microsoft.com/fwlink/?LinkId=131739).

See Also

Implementing Automated Testing

Using BizUnit to Facilitate Automated Testing

This section describes the stages of a BizUnit test case, how to implement a BizUnit XML configuration file, and how to use BizUnit in conjunction with LoadGen to automate testing.

In This Section

- Stages of a BizUnit Test Case
- Defining Testing Using an XML Configuration File
- Using BizUnit and LoadGen to Automate Performance and Stability Testing
- **Using BizUnit with the Business Process Management Scenario**

Stages of a BizUnit Test Case

Each BizUnit test case consists of three stages: **TestSetup**, **TestExecution**, and **TestCleanup**. Each stage contains one or more test steps that are responsible for performing a single discrete unit of work; for example, the **FileCreateStep** is responsible for creating a file in a location you specify with a given filename. BizUnit includes over 70 test steps and also provides extension capabilities which allow new test steps to be easily added to the framework. The ability to add new steps to the framework allows BizUnit to be used across a broad range of scenarios. This topic describes the BizUnit test stages in further detail.

Setup Stage

This setup stage prepares the platform for the testing. For example, before a particular test can be run, a file may need to be copied to a file drop in preparation for the actual execution of the test. You could also use this stage to cleanup any file locations or database tables that will be used in the test. As with every stage in BizUnit, there is no limit to the number of test steps that can be added, which provides the flexibility required to handle complex scenarios.

Execution Stage

The execution stage is where the test is actually run. This is where the function of the system you are validating is actually tested.

Cleanup Stage

The cleanup stage is the container for test steps that returns the platform to the consistent state that it was in before you ran the test. This stage is always executed, even if an error occurs in the execution stage. The reason the platform should be returned to its starting point is to prevent one

test case from interfering with another so that each test case is run autonomously as part of the test suite. Ensuring a complete cleanup of the system at this stage is one of the guiding principles for effective testing with BizUnit.

The following diagram illustrates the format of a sample test case, which contains test steps in the three stages: setup, execution, and cleanup. It is important to always follow this structure when defining test cases with BizUnit.

Stages of a BizUnit test

See Also

Using BizUnit to Facilitate Automated Testing

Defining Testing Using an XML Configuration File

BizUnit offers two ways to define tests: via an XML configuration file and via an Excel worksheet. This topic focuses on using an XML configuration file to define tests; however, you should also look at the BizUnit SDK, since it provides an interesting example of how to define BizUnit test cases using Excel. In addition, you may wish to investigate the BizUnit Designer tool, which provides a GUI that allows for rapid creation of BizUnit test cases. This topic provides an overview of how to define test cases using XML configuration using a very simplified scenario. If you are already familiar with BizUnit, you may want to skip this topic and proceed to the detailed end-to-end scenario described in **Walkthrough: Using BizUnit to Test the BPM Scenario**.

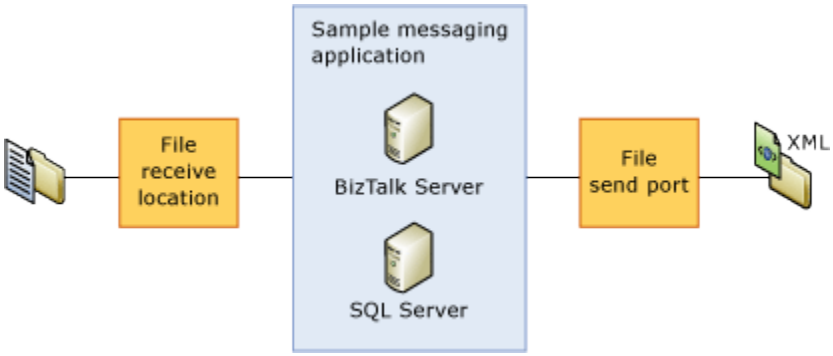
Overview of defining a BizUnit test case using XML configuration

As just stated, this scenario is simplified for purposes of illustration. Consider a sample messaging application such as the one illustrated below. Let’s assume that normal functional

behavior for this application is that BizTalk receives an XML file via a file receive location, it then sends it to an appropriate subscriber based on a subscription. To validate this scenario effectively it is important that we perform the following steps in the test:

1. Set up the environment to ensure that it is in a consistent state and ready for the test to be run:
   - This is done by deleting any files that are present in the two file locations used.
2. Run the test to verify functionality:
   - Create a valid XML message in the folder that the file receive location polls.
   - Validate that the correct XML message is placed in the outbound folder location.
   - The validation should cover both the schema and the payload information for the message. (Typically a couple of the key fields should be examined.)
3. Clean up the environment to ensure that the environment is in the same state as before the test executed:
   - Delete any files that are present in the two file locations used.

Sample BizTalk Messaging application



Each test case begins and ends with the TestCase XML tag; the testName parameter is passed into this as indicated here.

```
<TestCase testName="Test_01_FILECopyWithXmlValidation">
```

Then we enter the TestSetup phase, in which we ensure that the environment is in a consistent state to run the test. In this example, we delete any XML messages that are contained in our TestData directory. This is done using the **FileDeleteMultipleStep**.

```
   <TestSetup>
      <TestStep assemblyPath=""
typeName="Microsoft.Services.BizTalkApplicationFramework.BizUnit.FileDeleteMultipleStep">
         <Directory>..\..\..\TestData\</Directory>
            <SearchPattern>*.xml</SearchPattern>
         </TestStep>
   </TestSetup>
```

We then enter what is the most critical section of the test, the test execution stage. This stage can contain multiple test steps. In this example we use the FileCreateStep to copy a document (InDoc1.xml which can be seen in the <SourcePath> tag) to a file drop which is used by our receive location. It's important to note that BizUnit supports using unique identifiers for filenames in this step; this can be seen with the %Guid% reference in the CreationPath tag.

After this has been completed, we need to use the **FileValidateStep** to validate the outbound message has been created. You will notice this step allows you to specify a timeout value (this is in milliseconds), the directory and the search pattern. In addition to this, the **DeleteFile** tag enables you to specify whether you want the file to be removed after it has been validated. Finally, you should also note the validation includes an XPath query, which validates the PONumber node within the XML message (it checks that the value is PONumber_0.) Examination and validation of the payload of any outbound messages is another example of a guiding principle that you should follow when using BizUnit.

```
   <TestExecution>
      <TestStep assemblyPath=""
typeName="Microsoft.Services.BizTalkApplicationFramework.BizUnit.FileCreateStep">
         <SourcePath>..\..\..\TestData\InDoc1.xml</SourcePath>
         <CreationPath>..\..\..\Rec_03\TransactionId_%Guid%.xml</CreationPath>
      </TestStep>
      <TestStep assemblyPath=""
typeName="Microsoft.Services.BizTalkApplicationFramework.BizUnit.FileValidateStep">
         <Timeout>3000</Timeout>
         <Directory>..\..\..\Rec_03\</Directory>
         <SearchPattern>TransactionId_*.xml</SearchPattern>
         <DeleteFile>true</DeleteFile>
         <ValidationStep assemblyPath=""
typeName="Microsoft.Services.BizTalkApplicationFramework.BizUnit.XmlValidationStep">
            <XmlSchemaPath>..\..\..\TestData\PurchaseOrder.xsd</XmlSchemaPath>
            <XmlSchemaNameSpace>http://SendMail.PurchaseOrder</XmlSchemaNameSpace>
            <XPathList>
               <XPathValidation query="/*[local-name()='PurchaseOrder' and namespace-
uri()='http://SendMail.PurchaseOrder']/*[local-name()='PONumber' and namespace-
uri()='']">PONumber_0</XPathValidation>
            </XPathList>
         </ValidationStep>
      </TestStep>
   </TestExecution>
```

The final stage of the test case is the cleanup. As can be seen here, the **FileDelete** test step is used to clean up the directories used by the test.

```
   <TestCleanup>
      <TestStep assemblyPath=""
typeName="Microsoft.Services.BizTalkApplicationFramework.BizUnit.FileDeleteMultipleStep">
         <Directory>..\..\..\TestData\</Directory>
         <SearchPattern>*.xml</SearchPattern>
      </TestStep>
      <TestStep assemblyPath=""
typeName="Microsoft.Services.BizTalkApplicationFramework.BizUnit.FileDeleteMultipleStep">
         <Directory>..\..\..\Rec_03\</Directory>
         <SearchPattern>*.xml</SearchPattern>
      </TestStep>
   </TestCleanup>
```

Hopefully this example illustrates that defining tests in BizUnit is relatively straightforward and that by using this test framework, you will be able to rapidly develop test cases to provide functional testing of your application.

Full test case example

The full test case configuration file contents are included here for your reference:

```
<TestCase testName="Test_01_FILECopyWithXmlValidation">
   <TestSetup>
      <TestStep assemblyPath=""
typeName="Microsoft.Services.BizTalkApplicationFramework.BizUnit.FileDeleteMultipleStep">
         <Directory>..\..\..\TestData\</Directory>
            <SearchPattern>*.xml</SearchPattern>
         </TestStep>
   </TestSetup>
   <TestExecution>
      <TestStep assemblyPath=""
typeName="Microsoft.Services.BizTalkApplicationFramework.BizUnit.FileCreateStep">
         <SourcePath>..\..\..\TestData\InDoc1.xml</SourcePath>
         <CreationPath>..\..\..\Rec_03\TransactionId_%Guid%.xml</CreationPath>
      </TestStep>
      <TestStep assemblyPath=""
typeName="Microsoft.Services.BizTalkApplicationFramework.BizUnit.FileValidateStep">
```

```
         <Timeout>3000</Timeout>
         <Directory>..\..\..\Rec_03\</Directory>
         <SearchPattern>TransactionId_*.xml</SearchPattern>
         <DeleteFile>true</DeleteFile>
         <ValidationStep assemblyPath=""
typeName="Microsoft.Services.BizTalkApplicationFramework.BizUnit.XmlValidationStep">
            <XmlSchemaPath>..\..\..\TestData\PurchaseOrder.xsd</XmlSchemaPath>
            <XmlSchemaNameSpace>http://SendMail.PurchaseOrder</XmlSchemaNameSpace>
            <XPathList>
               <XPathValidation query="/*[local-name()='PurchaseOrder' and namespace-
uri()='http://SendMail.PurchaseOrder']/*[local-name()='PONumber' and namespace-
uri()='']">PONumber_0</XPathValidation>
            </XPathList>
         </ValidationStep>
      </TestStep>
   </TestExecution>
   <!-- Test cleanup: test cases should always leave the system in the state they found
it -->
   <TestCleanup>
      <TestStep assemblyPath=""
typeName="Microsoft.Services.BizTalkApplicationFramework.BizUnit.FileDeleteMultipleStep">
         <Directory>..\..\..\TestData\</Directory>
         <SearchPattern>*.xml</SearchPattern>
      </TestStep>
      <TestStep assemblyPath=""
typeName="Microsoft.Services.BizTalkApplicationFramework.BizUnit.FileDeleteMultipleStep">
         <Directory>..\..\..\Rec_03\</Directory>
         <SearchPattern>*.xml</SearchPattern>
      </TestStep>
   </TestCleanup>
</TestCase>
```

See Also

Using BizUnit to Facilitate Automated Testing

Using BizUnit and LoadGen to Automate Performance and Stability Testing

This topic provides information about how to use the Microsoft BizTalk LoadGen 2007 tool with BizUnit to automate performance and stability testing of a BizTalk Server solution.

BizTalk Server performance testing, step-by-step

Before investigating how to automate BizTalk Server performance testing, it is useful to know what steps are typically performed in a performance test. The following steps are representative of a “typical” BizTalk Server performance testing process:

1. Create a test results directory to store results and data collected, for example, event logs, trace logs, Performance Monitor data.
2. Clear the event logs on all servers within the test environment.
3. Stop all BizTalk host instances.
4. Stop any IIS instances that are running isolated BizTalk hosts such as the SOAP and HTTP receive adapter handlers.
5. Restart all instances of SQL Server used by the computers running BizTalk Server.
6. Clean up the MessageBox database to ensure that there is no leftover data from previous tests runs. This is important because any leftover data could skew test results.
7. Start the BizTalk host instances.
8. Start the relevant Performance Monitor counters on all servers in the test environment.
9. Send “priming” messages through the system to initialize the system caches.
10. After the priming messages have been processed, track the test start time in a SQL Server test results database.
11. Start the performance test by starting all of the LoadGen test agents.
12. Wait for the test to complete – often this can be done systematically by measuring the value of a Performance Monitor counter such as host queue length.
13. Track test end time in a SQL test results database.
14. Stop the relevant Performance Monitor counters on all servers in the test environment.
15. Save the test data to the test results directory that was created earlier.
16. Perform any necessary cleanup for the next test run.

Each and every one of the steps just listed can be automated using BizUnit. By utilizing the existing test step assets that BizUnit provides, you can quickly and easily generate an automated performance test for a BizTalk Server solution. One of the primary benefits of automating performance testing by using BizUnit is the flexibility to run tests overnight – meaning that the results are ready for analysis in the morning. This greatly reduces the burden of performance testing on a project team.

The Microsoft BizTalk LoadGen 2007 tool

The BizTalk LoadGen 2007 tool (LoadGen) is a load testing tool that was developed by the Stress and Performance Testing team in the BizTalk Server 2006 product group. LoadGen was designed to quickly, easily, and reliably define load tests that simulate production level message

volumes. LoadGen is multi-threaded, configuration-driven, and supports multiple transports. The BizTalk product group uses LoadGen on a daily basis; therefore you can have a high degree of confidence that the tool is durable, fit for the purpose, and able to simulate a wide variety of BizTalk scenarios.

LoadGen employs a modular design that consists of three layers: **presentation**, **framework** and **component**. The presentation layer consists of a command-line driver, which is responsible for driving the framework. The framework layer reads a configuration file and then executes the components specified therein. The component layer consists of three types of components: **load generators**, **message creators** and **throttle controllers**. Each of these components is extensible, so you can create your own and plug them into LoadGen to address the needs of your scenario. Because LoadGen was developed by the BizTalk Server product group, you should find that the out-of-the-box components will fulfill most of your load testing requirements. Each of these components is described in greater detail here.

- **Load generators** are responsible for transmitting messages via a particular transport. Load generators are provided for the following transports:
  - File
  - HTTP
  - MQSeries
  - MSMQLarge
  - MSMQ
  - SOAP
  - Web Services Enhancements (WSE)
  - Windows SharePoint Services (WSS)
  - Windows Communication Foundation (WCF)
- **Message creators** are an optional component that can be used when you need to generate messages that contain unique data. Message creators use one of two modes of creation; synchronous and asynchronous. If the synchronous message creation mode is specified, LoadGen uses only a single thread to create messages to ensure that each message contains a unique payload. While the synchronous mode guarantees unique data within each message, this mode also limits scalability. LoadGen also provides asynchronous message creators that use multiple execution threads; this enables LoadGen to meet the target message rate (because it can simply create more threads). In asynchronous mode, the message creator may be configured to randomly modify data for each individual message. However, because it uses multiple threads, it does not guarantee that all message generated during the test will contain a unique payload.
- **Throttle controllers** ensure that messages are transmitted at a steady rate by governing the load generators while the test is running. LoadGen also exposes custom throttling, which enables you to control the flow of messages based on criteria including:
  - Number of files in a folder
  - Number of rows in a database table
  - Depth of an MSMQ or MQSeries message queue

The Microsoft BizTalk LoadGen 2007 tool is available for download at http://go.microsoft.com/fwlink/?LinkId=59841 (http://go.microsoft.com/fwlink/?LinkId=59841).

Sample LoadGen configuration file

All LoadGen configuration information is stored in an xml file. The LoadGen configuration file contains a <CommonSection> element that configures the default settings for all LoadGen tasks in the LoadGen scenario. The LoadGen configuration file can also contain one or more <Section> elements that provide configuration settings for a specific LoadGen task. Entries in a <Section> element supersede any default values specified in the <CommonSection> element.

The sample LoadGen configuration file that follows is a slightly modified version of the FileToFileLG.xml sample configuration file that is included in the \ConfigFiles\ConsoleConfigFiles subdirectory of the LoadGen installation directory. This test sends 25 messages <LotSizePerInterval> every 200 milliseconds <SleepInterval>, 5 threads per load generator <NumThreadsperSection>and will stop the load test after 5000 messages <NumFiles> have been sent.

The file throttle controller is specified in the <ThrottleController> section. The value for <ThresholdRange> is set to 1000-2000, which means that if the file location C:\Scenarios\FileToFile\Receive (Parameters) has less than 1000 or more than 2000 files, the throttle controller will throttle the file generator and increase/decrease load as appropriate. The number of files in the file location will be checked every 1000 milliseconds <SleepInterval>. The <FileSection> element defines the properties for the messages to be sent by the load generators. The FileToFileLG.xml file <SrcFilePath> will be copied by LoadGen to the filedrop C:\Scenarios\FileToFile\Receive <DstFilePath>. The file transport is used here because this is the default transport specified in the <Transport Name> element within the <CommonSection> element.

```
<LoadGenFramework>

   <CommonSection>

      <LoadGenVersion>2</LoadGenVersion>

      <OptimizeLimitFileSize>204800</OptimizeLimitFileSize>

      <NumThreadsPerSection>5</NumThreadsPerSection>

      <SleepInterval>200</SleepInterval>

      <LotSizePerInterval>25</LotSizePerInterval>

      <RetryInterval>10000</RetryInterval>

      <StopMode Mode="Files">

         <NumFiles>5000</NumFiles>

      </StopMode>

      <Transport Name="FILE">

         <Assembly>FileTransport.dll/FileTransport.FileTransport</Assembly>

      </Transport>
```

```
      <ThrottleController Mode="Custom">
         <Monitor Name="File">

<Assembly>FileMonitor.dll/DropLocationFileMonitor.DropLocationFileMonitor</Assembly>
            <ThresholdRange>1000-2000</ThresholdRange>
            <SleepInterval>1000</SleepInterval>
            <Parameters>C:\Scenarios\FileToFile\Receive</Parameters>
         </Monitor>
         <ThrottleCondition>File</ThrottleCondition>
      </ThrottleController>
   </CommonSection>
   <Section Name="FileSection">

<SrcFilePath>C:\LoadGen\ConfigFiles\ConsoleConfigFiles\FileToFileLG.xml</SrcFilePath>
      <DstLocation>
         <Parameters>
            <DstFilePath>C:\Scenarios\FileToFile\Receive</DstFilePath>
         </Parameters>
      </DstLocation>
   </Section>
</LoadGenFramework>
```

Using BizUnit to drive LoadGen

BizUnit provides the **LoadGenExecuteStep** to facilitate automated performance and stability testing. The **TestExecution** stage of a sample BizUnit configuration file that uses **LoadGenExecuteStep** is shown in the following code example. Note that this step accepts a single configuration parameter, which is the location of the LoadGen configuration file.

```
<TestCase testName="Test_LoadGen">
   <TestSetup>
   </TestSetup>
   <TestExecution>
      <TestStep assemblyPath="" typeName="BizUnit.LoadGenExecuteStep,
BizUnit.LoadGenSteps">
         <LoadGenTestConfig>..\..\..\PerfGuideFiletoFile.xml</LoadGenTestConfig>
      </TestStep>
   </TestExecution>
```

```
   <!-- Test cleanup: test cases should always leave the system in the state they found
it -->
   <TestCleanup>
   </TestCleanup>
</TestCase>
```

The remainder of this topic describes the configuration file for a BizUnit test case that automates performance testing with LoadGen.

Note

This configuration file can be used as a template to quickly integrate BizUnit and LoadGen as part of your performance testing. Before running this test case, you will need to customize the configuration file for your environment. Sections of the configuration file that must be customized are indicated accordingly.

To begin with, specify a value for the **testName** parameter that is appropriate for the BizTalk solution.

```
<TestCase testName="Performance-Guide-Sample-Loadgen-Test">
```

Then add context variables to the **TestSetup** stage. These context variables will be referenced throughout the duration of the test case. To use this configuration file, modify the values specified for **TestCaseResultsDir** (C:\Dev Work\Perf Guide Demos\PerfResults\) and **Machine** (BIZTALKADMIN01) to match your environment.

```
<TestSetup>
   <!-- Context property: name of test run -->
   <TestStep assemblyPath="" typeName="BizUnit.ContextManipulatorStep">
      <ContextItem contextKey="TestRunName">
         <ItemTest takeFromCtx="BizUnitTestCaseName"></ItemTest>
         <ItemTest>_%DateTime%</ItemTest>
      </ContextItem>
   </TestStep>
   <!-- Context property: name of test directory to store results -->
   <TestStep assemblyPath="" typeName="BizUnit.ContextManipulatorStep">
      <ContextItem contextKey="TestCaseResultsDir">
         <ItemTest>C:\Dev Work\Perf Guide Demos\PerfResults\</ItemTest>
         <ItemTest takeFromCtx="TestRunName"></ItemTest>
      </ContextItem>
   </TestStep>
   <!-- Context property: perfmon log file -->
   <TestStep assemblyPath="" typeName="BizUnit.ContextManipulatorStep">
```

```
      <ContextItem contextKey="PerfMonFilePath">
         <ItemTest takeFromCtx="TestCaseResultsDir"></ItemTest>
         <ItemTest>\PerfCounters.blg</ItemTest>
      </ContextItem>
   </TestStep>
   <!-- Context property: destintation for app event log on test computer -->
   <TestStep assemblyPath="" typeName="BizUnit.ContextManipulatorStep">
      <ContextItem contextKey="DestPath- BIZTALKADMIN01-AppEventLog">
         <ItemTest takeFromCtx="TestCaseResultsDir"></ItemTest>
         <ItemTest> BIZTALKADMIN01_ApplicationLog.evt</ItemTest>
      </ContextItem>
   </TestStep>
   <!-- Clear the application event log on test computer -->
   <TestStep assemblyPath="" typeName="BizUnit.EventLogClearStep">
      <Machine>BIZTALKADMIN01</Machine>
      <EventLog>Application</EventLog>
   </TestStep>
   <!-- Create the directory to save all the test results -->
   <TestStep assemblyPath="" typeName="BizUnit.CreateDirectory">
      <DirectoryName takeFromCtx="TestCaseResultsDir" ></DirectoryName>
   </TestStep>
</TestSetup>
```

After completing the **TestSetup** stage, we enter the **TestExecution** stage. The first step is to stop the BizTalk host instances. A separate **BizUnit.HostConductorStep** section must be added for each distinct host instance. If you are using this configuration file in your environment, you will also need to enter the appropriate values for **HostInstanceName**, **Server**, **Logon**, and **Password**.

```
<TestExecution>
   <!-- Step 1: Stop BizTalk Hosts -->
   <TestStep assemblyPath="" typeName="BizUnit.HostConductorStep, BizUnit.BizTalkSteps">
      <Action>stop</Action>
      <HostInstanceName>BizTalkServerApplication</HostInstanceName>
      <Server>BizTalkAdmin01</Server>
      <Logon>ServerName\Administrator</Logon>
      <PassWord>Pass@word1</PassWord>
```

```
      <GrantLogOnAsService>true</GrantLogOnAsService>
   </TestStep>
```

After stopping all of the host instances, we clean up the BizTalk MessageBox database using the bts_CleanupMsgBox stored procedure. To use this step you must modify the value for **ConnectionString** to match your environment.

```
<!-- Step 2: Clean Up MessageBox -->
<TestStep assemblyPath="" typeName="BizUnit.DBExecuteNonQueryStep">
   <DelayBeforeExecution>1</DelayBeforeExecution>
   <ConnectionString>Persist Security Info=False;Integrated
Security=SSPI;database=BizTalkMsgBoxDb;server=BIZTALKADMIN01;Connect
Timeout=30</ConnectionString>
   <SQLQuery>
      <RawSQLQuery>[dbo].[bts_CleanupMsgbox]</RawSQLQuery>
   </SQLQuery>
</TestStep>
```

Step 3 of the **TestExecution** stage starts Performance Monitor (PerfMon) counters that are specified in a template file. A sample template file is listed underneath the sample **BizUnit.PerfmonCountersStep** below. To use the template file, you must modify the value specified for **CountersListFilePath** to match your environment. Modify the sample performance monitor counter template file to include any PerfMon counters that you would like to monitor or remove any that are not relevant to your scenario.

```
<!-- Step 3: Start Perfmon counters -->
<TestStep assemblyPath="" typeName="BizUnit.PerfmonCountersStep">
   <PerfmonAction>Start</PerfmonAction>
   <CounterSetName>PerfGuidePerfmonCounters</CounterSetName>
   <CountersListFilePath>C:\Dev Work\Perf Guide
Demos\Test_06_PerfCounters.txt</CountersListFilePath>
   <SampleInterval>5</SampleInterval>
   <PerfmonLogFilePath takeFromCtx="PerfMonFilePath"></PerfmonLogFilePath>
</TestStep>
```

Sample Performance Monitor counter template file (Test_06_PerfCounters.txt referenced by the BizUnit.PerfmonCountersStep):

```
\Processor(*)\*
\Process(*)\*
\Memory\*
\PhysicalDisk(*)\*
```

```
\System\Context Switches/sec

\System\Processor Queue Length

\BizTalk:FILE Receive Adapter(*)\*

\BizTalk:File Send Adapter(*)\*

\BizTalk:FTP Receive Adapter(*)\*

\BizTalk:FTP Send Adapter(*)\*

\BizTalk:HTTP Receive Adapter(*)\*

\BizTalk:HTTP Send Adapter(*)\*

\BizTalk:Message Agent(*)\*

\BizTalk:Messaging(*)\*

\BizTalk:Message Box:General Counters(*)\*

\BizTalk:Message Box:Host Counters(*)\*

\BizTalk:Messaging Latency(*)\*

\BizTalk:SOAP Receive Adapter(*)\*

\BizTalk:SOAP Send Adapter(*)\*

\BizTalk:TDDS(*)\*

\XLANG/s Orchestrations(*)\*
```

Now we start the BizTalk Server host instances. A separate **BizUnit.HostConductorStep** section must be added for each distinct host instance (distinct includes multiple instances of a host across servers). If you are using this configuration file in your environment, you will also need to enter the appropriate values for **HostInstanceName**, **Server**, **Logon**, and **Password**.

```
<!-- Step 4: Start BizTalk Hosts -->

<TestStep assemblyPath="" typeName="BizUnit.BizTalkSteps.HostConductorStep,

BizUnit.BizTalkSteps, Version=3.0.0.0, Culture=neutral, PublicKeyToken=7eb7d82981ae5162">

   <Action>start</Action>

   <HostInstanceName>BizTalkServerApplication</HostInstanceName>

   <Server>BizTalkAdmin01</Server>

   <Logon>ServerName\Administrator</Logon>

   <PassWord>Pass@word1</PassWord>

   <GrantLogOnAsService>true</GrantLogOnAsService>

</TestStep>
```

Step 5 “primes” the system by sending a couple of messages to BizTalk Server using **BizUnit.LoadGenExecuteStep**; change the value of the **LoadGenTestConfig** parameter to match your environment.

```
<!-- Step 5: Send Priming messages -->
```

```
<TestStep assemblyPath="" typeName="BizUnit.LoadGenExecuteStep, BizUnit.LoadGenSteps">
   <LoadGenTestConfig>C:\Program
Files\LoadGen\ConfigFiles\ConsoleConfigFiles\PerfGuideFiletoFile.xml</LoadGenTestConfig>
</TestStep>
```

Step 6 writes the LoadGen configuration file to memory so that it can then be written to the test results database when the test is complete.

```
<!-- Step 6: Read loadgen file into context variable -->
<TestStep assemblyPath="" typeName="BizUnit.FileReadAndLoadToContext">
   <FilePath>C:\Program
Files\LoadGen\ConfigFiles\ConsoleConfigFiles\PerfGuideFiletoFile.xml</FilePath>
   <ContextPropertyName>LoadGenFileContent</ContextPropertyName>
</TestStep>
```

Now we write the test start time to a test results database. Modify the **ConnectionString** and **RawSQLQuery** parameters to match your environment.

```
<!-- Step 7: Update test results DB with test start time -->
<TestStep assemblyPath="" typeName="BizUnit.DBExecuteNonQueryStep">
   <DelayBeforeExecution>1</DelayBeforeExecution>
   <ConnectionString>Persist Security Info=False;Integrated
Security=SSPI;database=TestResults;server=BizTalkAdmin01;Connect
Timeout=30</ConnectionString>
   <SQLQuery>
      <RawSQLQuery>INSERT INTO tblPerformanceResults (Test_ID, StartTime,LoadGenFile)
VALUES ('{0}',GetDate(),'{1}' )</RawSQLQuery>
      <SQLQueryParams>
         <SQLQueryParam takeFromCtx="TestRunName"></SQLQueryParam>
         <SQLQueryParam takeFromCtx="LoadGenFileContent"></SQLQueryParam>
      </SQLQueryParams>
   </SQLQuery>
</TestStep>
```

Step 8 is where the actual performance test is initiated using **BizUnit.LoadGenExecuteStep**. This step specifies the same LoadGen configuration file that was used in step 5, but you can specify any valid LoadGen configuration file here. **BizUnit.DelayStep** is used in step 9 to impose a 5-second delay to allow time for messages to start flowing through the system. Host queue length is calculated using **BizUnit.PerMonCounterMonitorStep**. When this parameter reaches a value of 1 as specified in step 10, the test is concluded. Change the values for the **InstanceName** and **Server** parameters to match the name of the host instance and server that you would like to monitor in your environment.

```
<!-- Step 8: LoadGen: Load actual perf test -->
<TestStep assemblyPath="" typeName="BizUnit.LoadGenSteps.LoadGenExecuteStep,
BizUnit.LoadGenSteps , Version=3.0.0.0, Culture=neutral,
PublicKeyToken=7eb7d82981ae5162">
   <LoadGenTestConfig>C:\Program
Files\LoadGen\ConfigFiles\ConsoleConfigFiles\PerfGuideFiletoFile.xml</LoadGenTestConfig>
</TestStep>
<!-- Step 9: Delay for 5 secs to allow msgs to start flowing -->
<TestStep assemblyPath="" typeName="BizUnit.DelayStep">
   <Delay>5000</Delay>
</TestStep>
<!-- Step 10: Wait for Orch Host Queue depth to reach one -->
<TestStep assemblyPath="" typeName="BizUnit.PerfMonCounterMonitorStep">
   <CategoryName>BizTalk:Message Box:Host Counters</CategoryName>
   <CounterName>Host Queue - Length</CounterName>
   <InstanceName>BizTalkServerApplication:biztalkmsgboxdb:BizTalkAdmin01</InstanceName>
   <Server>BizTalkAdmin01</Server>
   <CounterTargetValue>1</CounterTargetValue>
</TestStep>
```

At the conclusion of the test we use **BizUnit.DBExecuteNonQueryStep** to update the test results database. Completion of this step signifies the end of the test execution stage, as indicated by the closing </TestExecution> tag. Again, you must modify the **ConnectionString** and **RawSQLQuery** parameters to match your environment.

```
   <!-- Step 11: Update test results DB with test stop time -->
   <TestStep assemblyPath="" typeName="BizUnit.DBExecuteNonQueryStep">
      <DelayBeforeExecution>1</DelayBeforeExecution>
      <ConnectionString>Persist Security Info=False;Integrated
Security=SSPI;database=TestResults;server=BIZTALKADMIN01;Connect
Timeout=30</ConnectionString>
      <SQLQuery>
         <RawSQLQuery>UPDATE tblPerformanceResults SET EndTime = GetDate() WHERE Test_ID
= '{0}'</RawSQLQuery>
         <SQLQueryParams>
            <SQLQueryParam takeFromCtx="TestRunName"></SQLQueryParam>
         </SQLQueryParams>
```

```
      </SQLQuery>
   </TestStep>
</TestExecution>
```

Upon concluding the execution stage we enter the test cleanup stage. This stage uses **BizUnit.PerfmonCountersStep** to stop the Performance Monitor counters that were started earlier (in Step 3).

```
<TestCleanup>
      <!-- Return system to state prior to test -->
      <!-- Stop perfmon counters -->
      <TestStep assemblyPath="" typeName="BizUnit.PerfmonCountersStep"
failOnError="false">
         <PerfmonAction>Stop</PerfmonAction>
         <CounterSetName>PerfGuidePerfmonCounters</CounterSetName>
      </TestStep>
   </TestCleanup>
</TestCase>
```

This example illustrated how BizUnit can be combined with LoadGen to automate performance testing. The load test described by the BizUnit configuration file can be executed from Visual Studio's testing tools in the same manner as the functional testing. Adopting this approach enables you to centrally manage, administer, and collect data for your performance testing.

By using BizUnit and LoadGen in an automated approach, it is very easy to schedule multiple test runs to occur during off hours, which will provide ample test results for analysis during normal working hours. When automating performance testing, consider using LoadGen scripts that model different loads through the system, for example you may wish to simulate varying degrees (75%, 100% and 125%) of the expected production message volume. When performing load testing, it is especially important to test the overload or "bad day" scenario. Before placing the system into production, you should know what the maximum sustainable throughput (MST) is for each test case in the BizTalk Server environment. For more information about maximum sustainable performance, see [What Is Sustainable Performance?](http://go.microsoft.com/fwlink/?LinkID=132304) (http://go.microsoft.com/fwlink/?LinkID=132304) in the BizTalk Server 2009 documentation.

The complete BizUnit LoadGen sample configuration file

The following list contains the entire contents of the BizUnit configuration file referenced earlier.

```
<TestCase testName="Performance-Guide-Sample-Loadgen-Test">
   <TestSetup>
      <!-- Context property: name of test run -->
      <TestStep assemblyPath="" typeName="BizUnit.ContextManipulatorStep">
         <ContextItem contextKey="TestRunName">
```

```
      <ItemTest takeFromCtx="BizUnitTestCaseName"></ItemTest>
      <ItemTest>_%DateTime%</ItemTest>
   </ContextItem>
</TestStep>
<!-- Context property: name of test directory to store results -->
<TestStep assemblyPath="" typeName="BizUnit.ContextManipulatorStep">
   <ContextItem contextKey="TestCaseResultsDir">
      <ItemTest>C:\Dev Work\Perf Guide Demos\PerfResults\</ItemTest>
      <ItemTest takeFromCtx="TestRunName"></ItemTest>
   </ContextItem>
</TestStep>
<!-- Context property: perfmon log file -->
<TestStep assemblyPath="" typeName="BizUnit.ContextManipulatorStep">
   <ContextItem contextKey="PerfMonFilePath">
      <ItemTest takeFromCtx="TestCaseResultsDir"></ItemTest>
      <ItemTest>\PerfCounters.blg</ItemTest>
   </ContextItem>
</TestStep>
<!-- Context property: destintation for app event log on BTSSVR-001 -->
<TestStep assemblyPath="" typeName="BizUnit.ContextManipulatorStep">
   <ContextItem contextKey="DestPath-BTSSVR-001-AppEventLog">
      <ItemTest takeFromCtx="TestCaseResultsDir"></ItemTest>
      <ItemTest>BTSSVR-001_ApplicationLog.evt</ItemTest>
   </ContextItem>
</TestStep>
<!-- Clear the application event log on BTSSVR-001 -->
<TestStep assemblyPath="" typeName="BizUnit.EventLogClearStep">
   <Machine>BIZTALKADMIN01</Machine>
   <EventLog>Application</EventLog>
</TestStep>
<!-- Create the directory to save all the test results -->
<TestStep assemblyPath="" typeName="BizUnit.CreateDirectory">
   <DirectoryName takeFromCtx="TestCaseResultsDir" ></DirectoryName>
</TestStep>
```

```
   </TestSetup>

   <TestExecution>
      <!-- Step 1: Stop BizTalk Hosts -->
      <TestStep assemblyPath="" typeName="BizUnit.HostConductorStep,
BizUnit.BizTalkSteps">
         <Action>stop</Action>
         <HostInstanceName>BizTalkServerApplication</HostInstanceName>
         <Server>BizTalkAdmin01</Server>
         <Logon>ServerName\Administrator</Logon>
         <PassWord>Pass@word1</PassWord>
         <GrantLogOnAsService>true</GrantLogOnAsService>
      </TestStep>
      <!-- Step 2: Clean Up MessageBox -->
      <TestStep assemblyPath="" typeName="BizUnit.DBExecuteNonQueryStep">
         <DelayBeforeExecution>1</DelayBeforeExecution>
         <ConnectionString>Persist Security Info=False;Integrated
Security=SSPI;database=BizTalkMsgBoxDb;server=BIZTALKADMIN01;Connect
Timeout=30</ConnectionString>
         <SQLQuery>
            <RawSQLQuery>[dbo].[bts_CleanupMsgbox]</RawSQLQuery>
         </SQLQuery>
      </TestStep>
      <!-- Step 3: Start Perfmon counters -->
      <TestStep assemblyPath="" typeName="BizUnit.PerfmonCountersStep">
         <PerfmonAction>Start</PerfmonAction>
         <CounterSetName>PerfGuidePerfmonCounters</CounterSetName>
         <CountersListFilePath>C:\Dev Work\Perf Guide
Demos\Test_06_PerfCounters.txt</CountersListFilePath>
         <SampleInterval>5</SampleInterval>
         <PerfmonLogFilePath takeFromCtx="PerfMonFilePath"></PerfmonLogFilePath>
      </TestStep>
      <!-- Step 4: Start BizTalk Hosts -->
      <TestStep assemblyPath="" typeName="BizUnit.BizTalkSteps.HostConductorStep,
BizUnit.BizTalkSteps, Version=3.0.0.0, Culture=neutral, PublicKeyToken=7eb7d82981ae5162">
```

```
         <Action>start</Action>
         <HostInstanceName>BizTalkServerApplication</HostInstanceName>
         <Server>BizTalkAdmin01</Server>
         <Logon>ServerName\Administrator</Logon>
         <PassWord>Pass@word1</PassWord>
         <GrantLogOnAsService>true</GrantLogOnAsService>
      </TestStep>
      <!-- Step 5: Send Priming messages -->
      <TestStep assemblyPath="" typeName="BizUnit.LoadGenExecuteStep,
BizUnit.LoadGenSteps">
         <LoadGenTestConfig>C:\Program
Files\LoadGen\ConfigFiles\ConsoleConfigFiles\PerfGuideFiletoFile.xml</LoadGenTestConfig>
      </TestStep>
      <!-- Step 6: Read loadgen file into context variable -->
      <TestStep assemblyPath="" typeName="BizUnit.FileReadAndLoadToContext">
         <FilePath>C:\Program
Files\LoadGen\ConfigFiles\ConsoleConfigFiles\PerfGuideFiletoFile.xml</FilePath>
         <ContextPropertyName>LoadGenFileContent</ContextPropertyName>
      </TestStep>
      <!-- Step 7: Update test results DB with test start time -->
      <TestStep assemblyPath="" typeName="BizUnit.DBExecuteNonQueryStep">
         <DelayBeforeExecution>1</DelayBeforeExecution>
         <ConnectionString>Persist Security Info=False;Integrated
Security=SSPI;database=TestResults;server=BizTalkAdmin01;Connect
Timeout=30</ConnectionString>
         <SQLQuery>
            <RawSQLQuery>INSERT INTO tblPerformanceResults (Test_ID,
StartTime,LoadGenFile) VALUES ('{0}',GetDate(),'{1}' )</RawSQLQuery>
            <SQLQueryParams>
               <SQLQueryParam takeFromCtx="TestRunName"></SQLQueryParam>
               <SQLQueryParam takeFromCtx="LoadGenFileContent"></SQLQueryParam>
            </SQLQueryParams>
         </SQLQuery>
      </TestStep>
      <!-- Step 8: LoadGen: Load actual perf test -->
```

```
      <TestStep assemblyPath="" typeName="BizUnit.LoadGenSteps.LoadGenExecuteStep,
BizUnit.LoadGenSteps , Version=3.0.0.0, Culture=neutral,
PublicKeyToken=7eb7d82981ae5162">
        <LoadGenTestConfig>C:\Program
Files\LoadGen\ConfigFiles\ConsoleConfigFiles\PerfGuideFiletoFile.xml</LoadGenTestConfig>
      </TestStep>
      <!-- Step 9: Delay for 5 secs to allow msgs to start flowing -->
      <TestStep assemblyPath="" typeName="BizUnit.DelayStep">
         <Delay>5000</Delay>
      </TestStep>
      <!-- Step 10: Wait for Orch Host Queue depth to reach one -->
      <TestStep assemblyPath="" typeName="BizUnit.PerfMonCounterMonitorStep">
         <CategoryName>BizTalk:Message Box:Host Counters</CategoryName>
         <CounterName>Host Queue - Length</CounterName>

<InstanceName>BizTalkServerApplication:biztalkmsgboxdb:BizTalkAdmin01</InstanceName>
         <Server>BizTalkAdmin01</Server>
         <CounterTargetValue>1</CounterTargetValue>
      </TestStep>
      <!-- Step 11: Update test results DB with test stop time -->
      <TestStep assemblyPath="" typeName="BizUnit.DBExecuteNonQueryStep">
         <DelayBeforeExecution>1</DelayBeforeExecution>
         <ConnectionString>Persist Security Info=False;Integrated
Security=SSPI;database=TestResults;server=BIZTALKADMIN01;Connect
Timeout=30</ConnectionString>
         <SQLQuery>
            <RawSQLQuery>UPDATE tblPerformanceResults SET EndTime = GetDate() WHERE
Test_ID = '{0}'</RawSQLQuery>
            <SQLQueryParams>
               <SQLQueryParam takeFromCtx="TestRunName"></SQLQueryParam>
            </SQLQueryParams>
         </SQLQuery>
      </TestStep>
   </TestExecution>
```

```
   <TestCleanup>
      <!-- Return system to state prior to test -->
      <!-- Stop perfmon counters -->
      <TestStep assemblyPath="" typeName="BizUnit.PerfmonCountersStep"
failOnError="false">
         <PerfmonAction>Stop</PerfmonAction>
         <CounterSetName>PerfGuidePerfmonCounters</CounterSetName>
      </TestStep>
   </TestCleanup>
</TestCase>
```

See Also

Using BizUnit to Facilitate Automated Testing

Using Visual Studio to Facilitate Automated Testing

Visual Studio 2010 provides powerful load test functionality that can simulate a load profile of up to hundreds of users simultaneously accessing a server application. This load testing functionality provides real time metrics for selected key performance indicators as well as the ability to store these metrics in a database for future analysis. This section describes the use of Visual Studio Load testing to evaluate the performance of a BizTalk Server application.

In This Section

- Step 1: Create a Unit Test to Submit Documents to BizTalk Server
- Step 2: Configure Load Test Controller and Agent Computers
- Step 3: Create a Load Test to Perform Multiple Unit Tests Simultaneously
- Step 4: Configure BizTalk Server Environment for Load Testing
- Step 5: Perform Step Load Pattern Tests to Determine Maximum Sustainable Throughput
- Step 6: Perform Constant Load Pattern Tests to Verify Maximum Sustainable Throughput

Step 1: Create a Unit Test to Submit Documents to BizTalk Server

Computer application servers such as BizTalk Server are designed to perform particular tasks on behalf of users. These tasks are initiated as client requests sent to the application server as messages that conform to a standard that the application server understands, via a protocol that the application server understands. For example, clients may initiate processing of email by sending internet e-mail messages to an email server via the SMTP protocol. Likewise, web servers process client HTML or ASP requests, database servers process client SQL requests and BizTalk Server can process client messages formatted in compliance with multiple industry message standards using numerous industry standard protocols. The workload capacity of an application server is typically measured by the number of messages that the application server can process in a given time period. The workload capacity of BizTalk Server is likewise measured

as the average number of “documents received per second”, “documents processed per second” and/or “orchestrations completed per second” over an extended period of time, such as a busy workday or even a work week. Visual Studio 2010 load test functionality can simulate a load profile of up to hundreds of users simultaneously accessing a server application. This load testing functionality provides real time metrics for selected key performance indicators as well as the ability to store these metrics in a database for future analysis. This document desribes the use of Visual Studio Test projects for the purpose of load testing a BizTalk Server application, including how to create unit tests, how to create load tests and how to configure load tests to capture performance counter data required to determine the Maximum Sustainable Throughput (MST) of a BizTalk Server application.

Creating a Visual Studio Unit Test to Submit Documents to BizTalk Server

A Visual Studio Unit test references the Microsoft.VisualStudio.TestTools.UnitTesting (http://go.microsoft.com/fwlink/?LinkID=132293) namespace which supplies several classes that provide support for unit testing. Of particular importance, the UnitTesting (http://go.microsoft.com/fwlink/?LinkID=132293) namespace includes the Microsoft.VisualStudio.TestTools.UnitTesting.TestContext (http://go.microsoft.com/fwlink/?LinkID=208233) class to store information provided to Unit tests and the Microsoft.VisualStudio.TestTools.UnitTesting.TestMethodAttribute (http://go.microsoft.com/fwlink/?LinkID=208235) class is used to define Test methods. For purposes of load testing BizTalk Server, Test methods should specify the message to be loaded and the endpoint/URL to which the message should be sent. The endpoint/URL will then serve as the message entry point into BizTalk Server when a corresponding BizTalk receive location is created.

For purposes of illustration, the sample code in this topic describes Test methods which utilize the System.ServiceModel.ChannelFactory(TChannel) (http://go.microsoft.com/fwlink/?LinkID=208238) class to send messages to service endpoints that use the WCF net.tcp endpoint and are monitored by the BizTalk WCF-Custom receive adapter. Service endpoints for test messages are defined in the Application Configuration (app.config) file of the Test project.

For more information about Visual Studio Unit Tests see Anatomy of a Unit Test (http://go.microsoft.com/fwlink/?LinkID=208232).

Follow the steps in the sections below to create a Test project with a Unit Test to submit documents to one or more BizTalk Server computers. These steps were completed using Visual Studio 2010 Ultimate Edition.

Set Visual Studio 2010 Test Project Options

1. Launch Visual Studio 2010 Ultimate edition. Click **Start**, point to **All Programs**, point to **Microsoft Visual Studio 2010** and then click **Microsoft Visual Studio 2010**.
2. In Visual Studio 2010, click **Tools** and then click **Options** to display the **Options** dialog box.
3. Click to expand **Test Tools** and then click **Test Project** to display options for creation of new test projects.
4. Set the **Default test project language:** to **Visual C# test project**.

5. Under the option to **Select the files that will be added to each new test project, by default:** select **Visual C# test project**, and uncheck all of the test types for Visual C# test projects except for **Unit Test**.
6. Click **OK** to close the **Options** dialog box.

Create a new Visual Studio 2010 Solution with a Test Project

1. Create the folder **C:\Projects** on the Visual Studio 2010 Ultimate computer.
2. In Visual Studio 2010 click **File**, point to **New**, and click **Project** to display the **New Project** dialog box.
3. Under **Installed Templates** click to expand **Visual C#**, and click **Test**.
4. At the bottom of the **New Project** dialog box specify the following options:
   - **Name: BTSLoad**
   - **Location: C:\Projects**
   - **Solution name: LoadTest**
5. Ensure that the option to **Create directory for solution** is checked and then click **OK**.
6. Add a folder to the BTSLoad project; this folder will contain test messages to be submitted to BizTalk Server. In Solution Explorer, right-click the BTSLoad project, point to **Add**, and click **New Folder**. A folder icon with the highlighted text **NewFolder1** will appear under the BTSLoad project, type **TestMessages** to change the highlighted text and press the **Enter** key to create the folder C:\Projects\LoadTest\BTSLoad\TestMessages.

Update the Code in the Test Project and add an Application Configuration File to the Test Project

1. In Solution Explorer click to select **UnitTest1.cs** and replace the existing code with the following sample code listing:

```
#region Using Directives
using System;
using System.IO;
using System.Diagnostics;
using System.Text;
using System.Configuration;
using System.Collections.Generic;
using System.Linq;
using System.Xml;
using System.ServiceModel;
using System.ServiceModel.Channels;
using Microsoft.VisualStudio.TestTools.UnitTesting;
#endregion


namespace Microsoft.BizTalk.Samples
```

```
{
    [TestClass]
    public class BTSLoadTest
    {
        #region Constants
        private const int MaxBufferSize = 2097152;
        private const string Source = "BTS Load Test";
        private const string Star = "*";
        private const string TestMessageFolderParameter =
"testMessageFolder";
        private const string TestMessageFolderDefault =
@"C:\Projects\LoadTest\BTSLoad\TestMessages";
        private const string TestMessageFolderFormat = @"Test
Message Folder = {0}";
        private const string TestXmlDocument =
"testxmldocument.xml";
        #endregion

        #region Private Instance Fields
        private TestContext testContextInstance;
        #endregion

        #region Private ThreadStatic Fields
        [ThreadStatic]
        private static ChannelFactory<IRequestChannel>
channelFactory;
        [ThreadStatic]
        private static IRequestChannel channel = null;
        [ThreadStatic]
        private static byte[] buffer = null;
        #endregion

        #region Private Static Fields
        private static string testMessageFolder = null;
        #endregion
```

```
        #region Public Instance Constructor
        public BTSLoadTest()
        {
        }
        #endregion

        #region Public Static Constructor
        static BTSLoadTest()
        {
            try
            {
                testMessageFolder =
ConfigurationManager.AppSettings[TestMessageFolderParameter];
                if (string.IsNullOrEmpty(testMessageFolder))
                {
                    testMessageFolder =
TestMessageFolderDefault;
                }
            }
            catch (Exception ex)
            {
                Trace.WriteLine(ex.Message);
                EventLog.WriteEntry(Source, ex.Message,
EventLogEntryType.Error);
            }
        }
        #endregion

        #region Public Properties
        /// <summary>
        ///Gets or sets the test context which provides
        ///information about and functionality for the current
test run.
```

```
        ///</summary>
        public TestContext TestContext
        {
            get
            {
                return testContextInstance;
            }
            set
            {
                testContextInstance = value;
            }
        }
        #endregion

        #region Test Methods

        [TestMethod]
        public void BTSMessaging()
        {
            InvokeBizTalkReceiveLocation("BTSMessagingEP",
                                         testMessageFolder,
                                         TestXmlDocument,
MessageVersion.Default,
                                         SessionMode.Allowed);
        }

        [TestMethod]
        public void BTSMessaging2()
        {
            InvokeBizTalkReceiveLocation("BTSMessagingEP2",
                                         testMessageFolder,
                                         TestXmlDocument,
```

```
MessageVersion.Default,
                                        SessionMode.Allowed);
        }

        [TestMethod]
        public void BTSOrchestration()
        {
            InvokeBizTalkReceiveLocation("BTSOrchestrationEP",
                                        testMessageFolder,
                                        TestXmlDocument,

MessageVersion.Default,
                                        SessionMode.Allowed);
        }
        #endregion

        #region Helper Methods
        public void InvokeBizTalkReceiveLocation(string
endpointConfigurationName,
                                        string
requestMessageFolder,
                                        string
requestMessageName,
                                        MessageVersion
messageVersion,
                                        SessionMode
sessionMode)
        {
            XmlTextReader xmlTextReader = null;
            Message requestMessage = null;
            Message responseMessage = null;

            try
            {
```

```
                if (channel == null || channel.State !=
CommunicationState.Opened)
                {
                    channelFactory = new
ChannelFactory<IRequestChannel>(endpointConfigurationName);
                    channelFactory.Endpoint.Contract.SessionMode
= sessionMode;
                    channel = channelFactory.CreateChannel();
                }
                if (buffer == null)
                {
                    string path =
Path.Combine(requestMessageFolder, requestMessageName);

                    string message;
                    using (StreamReader reader = new
StreamReader(File.Open(path, FileMode.Open, FileAccess.Read,
FileShare.Read)))
                    {
                        message = reader.ReadToEnd();
                    }
                    buffer = Encoding.UTF8.GetBytes(message);
                }
                MemoryStream stream = new MemoryStream(buffer);
                xmlTextReader = new XmlTextReader(stream);
                requestMessage =
Message.CreateMessage(messageVersion, Star, xmlTextReader);
                TestContext.BeginTimer(requestMessageName);
                responseMessage =
channel.Request(requestMessage);
            }
            catch (FaultException ex)
            {
                HandleException(ex);
                throw;
```

```
            }
            catch (CommunicationException ex)
            {
                HandleException(ex);
                throw;
            }
            catch (TimeoutException ex)
            {
                HandleException(ex);
                throw;
            }
            catch (Exception ex)
            {
                HandleException(ex);
                throw;
            }
            finally
            {
                TestContext.EndTimer(requestMessageName);
                CloseObjects(xmlTextReader,
                             requestMessage,
                             responseMessage);
            }
        }

        private void HandleException(Exception ex)
        {
            try
            {
                Trace.WriteLine(ex.Message);
                EventLog.WriteEntry(Source, ex.Message,
EventLogEntryType.Error);
            }
            catch (Exception)
```

```
            {
            }
        }

        private void CloseObjects(XmlTextReader xmlTextReader,
                                  Message requestMessage,
                                  Message responseMessage)
        {
            try
            {
                if (xmlTextReader != null)
                {
                    xmlTextReader.Close();
                }
                if (requestMessage != null)
                {
                    requestMessage.Close();
                }
                if (responseMessage != null)
                {
                    responseMessage.Close();
                }
            }
            catch (Exception)
            {
            }
        }

        #endregion
    }
}
```

2. Add an Application Configuration file to the Test project:
   a. In Solution Explorer, right-click the BTSLoad project, point to **Add** and click **New item**.
   b. In the **Add New Item** dialog box, under **Installed Templates**, click **General**.

c. In the list of items that are displayed click to select **Application Configuration File** and then click **Add**.

d. In Solution Explorer select the app.config file and replace the contents of the app.config file with the sample code listing below:

> **Important**
> For each client endpoint defined in this file, *BizTalk Server Computer* is a placeholder for the actual name of the BizTalk Server computer(s) that you will perform load testing against.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <!-- Bindings used by client endpoints -->
    <bindings>
      <netTcpBinding>
        <binding name="netTcpBinding" closeTimeout="01:10:00"
openTimeout="01:10:00" receiveTimeout="01:10:00"
sendTimeout="01:10:00" transactionFlow="false"
transferMode="Buffered" transactionProtocol="OleTransactions"
hostNameComparisonMode="StrongWildcard" listenBacklog="100"
maxBufferPoolSize="1048576" maxBufferSize="10485760"
maxConnections="400" maxReceivedMessageSize="10485760">
          <readerQuotas maxDepth="32"
maxStringContentLength="8192" maxArrayLength="16384"
maxBytesPerRead="4096" maxNameTableCharCount="16384" />
          <reliableSession ordered="true"
inactivityTimeout="00:10:00" enabled="false" />
          <security mode="None">
            <transport clientCredentialType="Windows"
protectionLevel="EncryptAndSign" />
            <message clientCredentialType="Windows" />
          </security>
        </binding>
      </netTcpBinding>
    </bindings>
    <client>
```

```
      <!-- Client endpoints used to exchange messages with WCF
Receive Locations -->

      <!-- BTSMessagingEP -->
      <endpoint address="net.tcp://BizTalk Server
Computer:8123/btsloadtest" binding="netTcpBinding"
bindingConfiguration="netTcpBinding"
contract="System.ServiceModel.Channels.IRequestChannel"
name="BTSMessagingEP" />
      <endpoint address="net.tcp://BizTalk Server
Computer:8123/btsloadtest" binding="netTcpBinding"
bindingConfiguration="netTcpBinding"
contract="System.ServiceModel.Channels.IRequestChannel"
name="BTSMessagingEP" />

      <!-- BTSOrchestrationEP -->
      <endpoint address="net.tcp://BizTalk Server
Computer:8122/btsloadtest" binding="netTcpBinding"
bindingConfiguration="netTcpBinding"
contract="System.ServiceModel.Channels.IRequestChannel"
name="BTSOrchestrationEP" />
    </client>
  </system.serviceModel>
  <appSettings>
    <!-- Folder containing test messages -->
    <add key="testMessageFolder"
value="C:\Projects\LoadTest\BTSLoad\TestMessages" />
    <add key="ClientSettingsProvider.ServiceUri" value="" />
  </appSettings>
</configuration>
```

e. Click the **File** menu in Visual Studio 2010 and then click **Save All**.

Add a Test Message to the Project

For purposes of this example, BizTalk Server receive location(s) and send port(s) will be configured to use pass through pipelines and will not perform any document validation. Follow these steps to add a test message to the project:

1. Launch Notepad. Click **Start**, click **Run** and type **Notepad** in the **Run** dialog box.

2. Copy the following text into Notepad and save as "C:\Projects\LoadTest\BTSLoad\TestMessages\TestXmlDocument.xml"

```
<BTSLoadTest
xmlns="http://Microsoft.BizTalk.Samples.BTSLoadTest">
<MessageText>
This is sample message text. This is sample message text. This
is sample message text. This is sample message text.
This is sample message text. This is sample message text. This
is sample message text. This is sample message text.
This is sample message text. This is sample message text. This
is sample message text. This is sample message text.
This is sample message text. This is sample message text. This
is sample message text. This is sample message text.
This is sample message text. This is sample message text. This
is sample message text. This is sample message text.
This is sample message text. This is sample message text. This
is sample message text. This is sample message text.
This is sample message text. This is sample message text. This
is sample message text. This is sample message text.
This is sample message text. This is sample message text. This
is sample message text. This is sample message text.
This is sample message text. This is sample message text. This
is sample message text. This is sample message text.
This is sample message text. This is sample message text. This
is sample message text. This is sample message text.
This is sample message text. This is sample message text. This
is sample message text. This is sample message text.
This is sample message text. This is sample message text. This
is sample message text. This is sample message text.
This is sample message text. This is sample message text. This
is sample message text. This is sample message text.
This is sample message text. This is sample message text. This
is sample message text. This is sample message text.
This is sample message text. This is sample message text. This
is sample message text. This is sample message text.
</MessageText>
</BTSLoadTest>
```

3. Close Notepad.

Important
This file will need to be saved to the same path using the same file name on every Load Test Agent computer if multiple Load Test Agent computers are used for load testing.

Add Necessary References to the Project and Build the Test Project

1. In Solution Explorer, right-click the **References** folder for the BTSLoad project and then click **Add Reference**.
2. In the Add Reference dialog box, click the **.NET** tab and use the CTRL+Click keyboard/mouse combination to simultaneously select the following .NET namespaces:
   - System.Configuration
   - System.Runtime.Serialization
   - System.ServiceModel.Channels
   - System.ServiceModel
   - System.Web.Extensions
   - System.Xml
3. After selecting the namespaces click **OK** to add these assemblies as references to the BTSLoad Test project.
4. Right click the **BTSLoad** project and then click **Build** to compile the project into the BTSLoad assembly.

Step 2: Configure Load Test Controller and Agent Computers

Visual Studio 2010 Ultimate edition can generate load simulating up to 250 virtual users on a local load test run. To simulate more than 250 virtual users and/or to initiate testing from a remote computer requires Visual Studio Load Test Virtual User Pack 2010.

All load testing performed for this guide was initiated from two computers:

- One computer running as both a Load Test Controller and a Load Test Agent.
- Another computer running as a Load Test Agent only.

Test results were stored in a remote load test results repository in a SQL Server 2008 R2 database.

For more information about using test controllers and test agents to distribute load tests across multiple test machines, see Distributing Load Tests Across Multiple Test Machines Using Test Controllers and Test Agents (http://go.microsoft.com/fwlink/?LinkId=208406).

Install and Configure the Load Test Controller and Load Test Agents

To install and configure the load test controller and load test agents, see the following sections in the topic Installing and Configuring Visual Studio Agents and Test and Build Controllers (http://go.microsoft.com/fwlink/?LinkId=208455):

- To setup a test controller, follow the procedures in the Install a Test Controller (http://go.microsoft.com/fwlink/?LinkId=208454) section.
- To setup test agents, follow the procedures in the Install a Test Agent (http://go.microsoft.com/fwlink/?LinkId=208456) section.

Step 3: Create a Load Test to Perform Multiple Unit Tests Simultaneously

Load tests run multiple instances of one or more unit tests so that you can measure your application's performance and ability to handle load. The primary components of a Visual Studio 2010 load test include:

- **Scenarios** – The section of a load test where you configure the test load pattern, test mix model, test mix, network mix and Web browser mix. Scenarios accommodate the complexity of simulating complex real world work load profiles. For a comprehensive listing of all load test scenario properties see Load Test Scenario Properties (http://go.microsoft.com/fwlink/?LinkId=208327).
- **Counter Sets** – The section of a load test where you create particular groupings or "Sets" of performance counters to be collected while the load test is running. Several predefined counter sets are provided by default and custom counter sets can be added. For example to evaluate Network performance you can create a custom counter set, add the relevant Network performance counters and save it to the list of available counter sets. For more information about creating and saving counter sets for load tests see Specifying the Counter Sets for Computers in a Load Test (http://go.microsoft.com/fwlink/?LinkId=208328).
- **Run Settings** – Run settings define multiple aspects of a load test, including the test duration, the counter sets that are associated with various computers during the load test, various test validation options, and test results storage options. You can create and store multiple run settings for each load test, and then select a particular setting to use when you run the test. An initial run setting is added to your load test when you create your load test with the New Load Test Wizard. For a comprehensive listing of all load test run setting properties see Load Test Run Setting Properties (http://go.microsoft.com/fwlink/?LinkId=208329).

Load tests are created using the New Load Test Wizard, edited with the Load Test Editor, and analyzed in the Load Test Analyzer. All these tools are included in Microsoft Visual Studio Ultimate edition. For more information about creating and editing load tests in Visual Studio 2010 Ultimate edition see Creating and Editing Load Tests (http://go.microsoft.com/fwlink/?LinkId=208308).

Follow the steps in the sections below to add a load test to the test project described in Step 1: Create a Unit Test to Submit Documents to BizTalk Server. These steps also describe how to configure the **Scenarios**, **Counter Sets**, and **Run Settings** for a load test.

Add a Load Test and Configure the Load Test Scenario, Counter Sets, and Run Settings

This topic describes how to use the **New Load Test Wizard** to add a load test to a test project and how to configure the load test to meet specific needs.

Use the New Load Test Wizard to Add a Load Test to the Test Project

Follow these steps to add a load test to a test project using the New Load Test Wizard.

1. Open the **Load Test** solution in Visual Studio 2010 if it is not already open.
2. Add a folder to the BTSLoad project; this folder will contain any load tests that are created as part of this project. In Solution Explorer, right-click the BTSLoad project, point to **Add**, and click **New Folder**. A folder icon with the highlighted text **NewFolder1** will appear under the

BTSLoad project, type **LoadTests** to change the highlighted text and press the Enter key to complete creation of the folder C:\Projects\LoadTest\BTSLoad\LoadTests.

3. In Solution Explorer, right-click on the **BTSLoad** project, point to **Add**, and then click **Load Test** to start the **New Load Test Wizard**.
4. Click **Next**.
5. On the **Edit Settings for a Load Test Scenario** page under **Enter a name for the load test scenario:** type **BTS_Messaging_Step**. Under **Think time profile** select **Do not use think times** and then click **Next**.
6. On the **Edit load pattern settings for a load test scenario** page select **Step load**, enter the values below and then click **Next**.
   - **Start user count:** 30 users
   - **Step duration:** 60 seconds
   - **Step user count:** 10 users
   - **Maximum user count** 80 users

   Note

   When applying settings for a step load pattern you should calculate the amount of time required for all step increments to complete. For example, using the load pattern settings specified above the load test will need 5 minutes to complete all of the 60 second step increments when ramping up from 30 to 80 users. On the last page of the New Load Test Wizard you will be presented with options for specifying the length of the load test, one of which will be **Load Test Duration**. If you have already calculated the time required for all step increments to complete then it is a straightforward task to enter the value (5 minutes in this case) for **Load Test Duration**.
7. On the **Select a test mix model for the load test** page select **Based on the number of virtual users** and then click **Next**.
8. On the **Add tests to the load test scenario and edit the test mix** page click the **Add** button.
9. Under **Available tests** double-click **BTSMessaging** and **BTSMessaging2** to add these unit tests to the list of **Selected tests**. Click **OK** and then click **Next**.
10. On the **Add network types to a load test scenario and edit the network mix** page verify that **Network Type** is set to **LAN** with a **Distribution** of **100%** and then click **Next**.
11. On the **Specify computers to monitor with counter sets during load test run** page click **Next**.

    Note

    Do not add computers to the load test at this time. The New Load Test Wizard will only allow you to associate computers with predefined counter sets, and this load test requires the use of both predefined and *custom* counter sets. After the wizard is complete and the load test is saved you can edit the load test to add custom counter sets and configure the load test to monitor computers using both predefined *and* custom counter sets.

On the **Review and edit run settings for a load test** page enter the following values:

a. Select **Load test duration**.

b. **Warm-up duration (hh mm ss)** 30 seconds

c. **Run duration (hh mm ss)** 5 minutes

> **Note**
> The time allocated for **Run duration** should equal the amount of time required for all step increments to complete as described in step 5 above, or 5 minutes for this example.

d. **Sampling rate** 5 seconds

e. **Description** (optional), Enter a description for the load test here.

f. **Save Log on Test Failure** True

g. **Validation level** Low – invoke validation rules marked low

12. Click **Finish** to close the New Load Test Wizard.
13. Click the **File** menu and select **Save <Load Test Name>.loadtest As**.

> **Note**
> In this example, <Load Test Name> will be the name assigned to the load test file by Visual Studio 2010, typically loadtestx.loadtest, unless the name of the file has already been manually changed.

14. Save the file to the C:\Projects\LoadTest\BTSLoad\LoadTests directory created earlier. It may be useful to save the file with the name used for the scenario; in this example the scenario name is BTS_Messaging_Step so the loadtest file would be saved as C:\Projects\LoadTest\BTSLoad\LoadTests\BTS_Messaging_Step.loadtest.

Add a Custom Counter Set to Measure BizTalk Server Key Performance Indicators (KPI)

Follow these steps to add a counter set with performance counters that measure BizTalk Server KPI required for determining Maximum Sustainable Throughput (MST) of the BizTalk Server application:

1. In Solution Explorer double-click the load test that you created in the previous section to view the load test in Load Test editor.
2. In Load Test editor, click to expand **Counter Sets**. Notice that there is no predefined counter set for BizTalk Server, therefore a custom “BizTalk Server” counter set must be added to the list of counter sets.
3. Right-click **Counter Sets** and select **Add Custom Counter Set**. By default this action will create a custom counter set with the name **Custom1**.
4. Right-click the **Custom1** counter set and select **Properties** to set focus to the **Properties** dialog for the Custom1 counter set.
5. Double-click the name **Custom1** in the **Properties** dialog, type **BizTalk** and then press the ENTER key to rename the custom counter set to **BizTalk**.
6. In Load Test Editor, right-click the **BizTalk** counter set and select **Add Counters**.

7. Under **Computer**, type the name of one of the BizTalk Server computers in the BizTalk Server group to display performance monitor categories that include BizTalk Server performance counters.

   Important
   To ensure that all BizTalk Server performance categories and performance counters are listed, you may need to type in the fully qualified domain name (or IP Address) of a BizTalk Server in the group and you may also need to start the instances of the following hosts on the BizTalk Server computer.

   - Instances of BizTalk hosts which are bound to orchestrations that will run during the load test.
   - Instances of BizTalk hosts configured as send or receive handlers for adapters that will run during the load test.

8. BizTalk Server provides quite an extensive set of performance counters. For purposes of determining the Maximum Sustainable Performance (MST) of a BizTalk Server application you only need to add the following BizTalk Server performance counters to the **BizTalk** custom counter set:

Performance Category	Performance Counter
Processor	% Processor Time for the _Total counter instance.
BizTalk:Message Box: General Counters	Spool Size for the *<BizTalk MessageBox database name>*:*<SQL Server instance name>* counter instance. **Note** *<BizTalk MessageBox database name>* and *<SQL Server instance name>* are just placeholders for the actual names of the BizTalk MessageBox database and the SQL Server instance that houses the BizTalk MessageBox database. These placeholders should be replaced with the actual names of the BizTalk MessageBox database and associated SQL Server instance.
BizTalk:Messaging	Documents received/Sec for the receive host counter instance. Documents processed/Sec for the transmit host counter instance.
BizTalk:Message Agent	Message delivery incoming rate for the document receive host.
BizTalk:Message Agent	Message publishing outgoing rate for the document transmit host.
XLANG/s Orchestrations	Orchestrations completed/sec for the Orchestration processing host.

Modify Run Settings to Map Counter Sets to Appropriate Computers

Follow these steps to map the appropriate counter sets with the appropriate computers for the load test:

1. In **Load Test Editor**, right-click **Run Settings** and select **Manage Counter Sets**.
2. Click **Add Computer** to add a new computer to the list. An icon with the highlighted text **New Computer** will appear under **Computers and counter sets to monitor**. Replace the highlighted text by typing the name of the computer you would like to add to the list.

3. After adding the computer to the list, click to expand the list of available counter sets and then click to select one or more of the available counter sets to associate the counter set(s) with the computer.
4. Repeat steps 2 and 3 until you have associated counter sets with all computers for which you would like to collect performance data.

Add a Test Settings File to the Solution to Run Tests and Collect Data Remotely

To configure the Load test to use the Test Controller and Test Agent computers that you created in Step 2: Configure Load Test Controller and Agent Computers, follow the steps in Add a test settings for remote execution or data collection to your solution(http://go.microsoft.com/fwlink/?LinkId=209182) as noted below:

1. For Step 3, enter the name **BizTalkLoadTest**
2. Ignore Step 6 because you have already entered a name in Step 3.
3. For Step 7, enter “These are default test settings for a remote test run” under **Description**.
4. For Step 8, select the default naming scheme.
5. For Step 9, under **Test execution method** select **Remote execution**, under **Controller** select the test controller computer and leave the remaining properties on the **Roles** page at their default settings.
6. For Step 24, select the option **to Run in default host**, select a **Host type** of **Default**, and under **Run tests in 32 or 64 bit process**, select the option to **Run tests in 64 bit process on 64 bit machine**.
7. For Step 25, select **Mark an individual test as failed if its execution time exceeds** and leave the default value of 30 minutes selected.
8. For Step 27b, select the check box for **Use the Load Context for assemblies in the test directory**, and then click **Save As**.
9. In the **Save As** dialog box, verify that the name **BizTalkLoadTest** is entered next to **File name**, and click **Save**. You have now added a test settings file to your solution.

Step 4: Configure BizTalk Server Environment for Load Testing

This topic provides information for creating the BizTalk Server Receive Locations, Receive Ports, and Send Ports required to run the sample code described in the topics Step 1: Create a Unit Test to Submit Documents to BizTalk Server and Step 3: Create a Load Test to Perform Multiple Unit Tests Simultaneously.

Configure BizTalk Server Environment for Load Tests

As described in the topic Step 3: Create a Load Test to Perform Multiple Unit Tests Simultaneously, the load test **BTS_Messaging_Step** is configured to execute the unit tests **BTSMessaging** and **BTSMessaging2**. In turn, these unit tests load a copy of the message C:\Projects\LoadTest\BTSLoad\TestMessages\TestXmlDocument.xml and send it to the endpoints **BTSMessagingEP** and **BTSMessagingEP2** as defined in the following section of the project’s application configuration (app.config) file:

```
<!-- BTSMessagingEP -->
```

```
      <endpoint address="net.tcp://BizTalk Server Computer:8123/btsloadtest"
binding="netTcpBinding" bindingConfiguration="netTcpBinding"
contract="System.ServiceModel.Channels.IRequestChannel" name="BTSMessagingEP" />
      <endpoint address="net.tcp://BizTalk Server Computer:8123/btsloadtest"
binding="netTcpBinding" bindingConfiguration="netTcpBinding"
contract="System.ServiceModel.Channels.IRequestChannel" name="BTSMessagingEP2" />
```

Note

As noted earlier, *BizTalk Server Computer* is a placeholder for the actual BizTalk Server computer names or, in the case that the BizTalk Server Computers are configured as members of a Network Load Balancing (NLB) cluster; *BizTalk Server Computer* is a placeholder for the name or address of the corresponding NLB virtual server.

For purposes of this example, two BizTalk Server computers were used and the BizTalk Server Message Box database was on a remote SQL Server computer.

Create BizTalk Server Send and Receive Hosts

Follow the steps in the BizTalk Server Documentation topic How to Create a New Host (http://go.microsoft.com/fwlink/?LinkId=208595) to create a BizTalk Server “Send” host for Send Ports and Send adapter handlers. Configure the host with the following properties:

Property	Value
Name	TxHost
Type	In-Process
Allow Host Tracking	Ensure that this box is un-checked.
Authentication Trusted	Ensure that this box is un-checked.
32-bit only	Ensure that this box is un-checked.
Make this the default host in the group	Ensure that this box is un-checked.
Windows group	The Windows group used to control access to this host and associated host instances. The Window group created for the default in-process host is named either <*Computer Name*>\BizTalk Application Users (for a single server BizTalk Server installation) or <*Domain Name*>\BizTalk Application Users (for a multiple server BizTalk Server installation, which requires the use of domain groups). **Note** <*Computer Name*> and <*Domain Name*> are placeholders for the actual

Property	Value
	computer name or domain name used when the group was created. If a new group is created for this host then it must have the privileges described in the topic Host Groups (http://go.microsoft.com/fwlink/?LinkId=208803) in the BizTalk Server documentation.

Repeat the steps that you followed when creating the “Send” host to create a “Receive” host. Configure the “Receive” host with the following property values:

Property	Value
Name	RxHost
Type	In-Process
Allow Host Tracking	Ensure that this box is un-checked.
Authentication Trusted	Ensure that this box is un-checked.
32-bit only	Ensure that this box is un-checked.
Make this the default host in the group	Ensure that this box is un-checked.
Windows group	The Windows group used to control access to this host and associated host instances. The Window group created for the default in-process host is named either <*Computer Name*>\BizTalk Application Users (for a single server BizTalk Server installation) or <*Domain Name*>\BizTalk Application Users (for a multiple server BizTalk Server installation, which requires the use of domain groups). **Note** <*Computer Name*> and <*Domain Name*> are placeholders for the actual computer name or domain name used when the group was created. If a new group is created for this host then it must have the privileges described in the topic Host Groups (http://go.microsoft.com/fwlink/?LinkId=208803)

Property	Value
	in the BizTalk Server documentation.

Create Instances of the BizTalk Server Send and Receive Hosts

Follow the steps in the BizTalk Server Documentation topic How to Add a Host Instance (http://go.microsoft.com/fwlink/?LinkId=208596) to create and start instances of the BizTalk Server “Send” host. Configure an instance of the “Send” host to run on each BizTalk Server in the BizTalk Server group and configure each host instance with the following property values:

Property	Value
Host name	Select **TxHost** from the drop-down list next to **Host name**.
Server	Select the BizTalk Server that will run this host instance from the drop-down list next to **Server**.
Logon	1. Click the **Configure** button to display the **Logon Credentials** dialog box. 2. In the **Logon Credentials** dialog box enter the following values for the specified properties: **Property** / **Value** Logon — Name of user account that is a member of the Windows group associated with this BizTalk Server host. Password — Password for the user account specified in the **Logon** textbox. 3. Click **OK** to close the **Logon Credentials** dialog box.
Disable host instance from starting.	Ensure that this box is un-checked.

After creating the host instance, right-click on the host instance and select **Start** from the context menu.

Repeat the steps that you followed when creating the “Send” host instances to create “Receive” host instances. Configure an instance of the “Receive” host to run on each BizTalk Server in the BizTalk Server group and configure each host instance with the following property values:

<table>
<tr><th>Property</th><th>Value</th></tr>
<tr><td>Host name</td><td>Select RxHost from the drop-down list next to Host name.</td></tr>
<tr><td>Server</td><td>Select the BizTalk Server that will run this host instance from the drop-down list next to Server.</td></tr>
<tr><td>Logon</td><td>1. Click the Configure button to display the Logon Credentials dialog box.
2. In the Logon Credentials dialog box enter the following values for the specified properties:
<table><tr><th>Property</th><th>Value</th></tr><tr><td>Logon</td><td>Name of user account that is a member of the Windows group associated with this BizTalk Server host.</td></tr><tr><td>Password</td><td>Password for the user account specified in the Logon textbox.</td></tr></table>3. Click OK to close the Logon Credentials dialog box.</td></tr>
<tr><td>Disable host instance from starting</td><td>Ensure that this box is un-checked.</td></tr>
</table>

After creating the host instance, right-click on the host instance and select **Start** from the context menu.

Create a BizTalk Server Receive Port

Follow the steps in the topic How to Create a Receive Port (http://go.microsoft.com/fwlink/?LinkID=154843) in the BizTalk Server documentation to create a

One-Way Receive Port. When creating the Receive port, leave all properties at default values except as noted in the table below:

Property	Value
General\Name	BTSLoadTestMessaging.OneWay.ReceivePort
General\Port Type	One-Way
General\Authentication	No authentication
General\Enable routing for failed messages	Ensure that this box is un-checked.
General\Description	Leave blank
Inbound Maps	None
Tracking	Ensure that all boxes are un-checked.
Receive Locations	Click **New**, this will display the **Receive Location Properties** dialog box which should be configured as described in the following section, **Create a BizTalk Server Receive Location**.

Create a BizTalk Server Receive Location

In the **Receive Location Properties** dialog box displayed while creating the BizTalk Server Receive port, apply the specified property values:

Property	Value
Name:	BTSLoadTest.Messaging.OneWay.WCF-Customer.ReceiveLocation
Receive handler:	RxHost
Receive pipeline:	PassThruReceive
Description:	Leave this blank
Type:	Select **WCF-Custom** from the drop-down list and then click the **Configure** button, this will display the **WCF-Custom Transport Properties** dialog box which should be configured as described in the following section, **Configure the WCF-Custom Receive Transport**.

Configure the WCF-Custom Receive Transport

In the **WCF-Custom Transport Properties** dialog box displayed while creating the BizTalk Server Receive location, leave all properties at default values except as noted in the table below:

Property	Value
General\Address (URI)	net.tcp://localhost:8123/btsloadtest
Binding\Binding Type	netTcpbinding
Binding\NetTcpBindingElement\listenBacklog	400
Binding\NetTcpBindingElement\maxConnections	400
Binding\Security\NetTcpSecurityElement\mode	None
Behavior\ServiceBehavior\serviceThrottling\ServiceThrottlingElement **Note** To add the serviceThrottling behavior to the list of behaviors, right-click ServiceBehavior, click **Add Extension**, select **serviceThrottling** from the list of behavior extensions and then click **OK**.	Set the **ServiceThrottlingElement** properties to the following values: • **maxConcurrentCalls** 400 • **maxConcurrentInstances** 400 • **maxConcurrentSessions** 400
Behavior\ServiceBehavior\serviceDebug\ServiceDebugElement **Note** To add the serviceDebug behavior to the list of behaviors, right-click ServiceBehavior, click **Add Extension**, select **serviceDebug** from the list of behavior extensions, and then click **OK**.	Leave the list of **ServiceDebugElement** properties at their default values (empty) except for the following properties, which should be changed to a value of True: • **httpHelpPageEnabled** True • **httpsHelpPageEnabled** True • **includeExceptionDetailInFaults** True

Click **OK** to close the WCF-Custom Transport Properties dialog box and then click **OK** again to close the Receive Location Properties dialog box.

Create a BizTalk Server Send Port

Follow the steps in the topic How to Create a Send Port (http://go.microsoft.com/fwlink/?LinkID=154845) in the BizTalk Server documentation to create a **Static One-Way** Send Port. When creating the Send port, leave all properties at default values except as noted in the table below:

Property	Value
General\Name	BTSLoadTest.Messaging.Send.WCF-Custom
General\Send handler	TxHost
General\Send pipeline	PassThruTransmit
Filters\Name	BTS.ReceivePortName
Filters\Operator	==
Filters\Value	BTSLoadTest.Messaging.OneWay.ReceivePort
Filters\Group by	And  **Note** If these properties are configured with the correct values, the filter should be displayed as `BTS.ReceivePortName == BTSLoadTest.Messaging.OneWay.ReceivePort` as seen toward the bottom of the Filters page of the Send Port Properties dialog box. As a result of applying this filter, this Send port subscribes to any messages received by BizTalk Server via the Receive Port named BTSLoadTest.Messaging.OneWay.ReceivePort.
Tracking	Ensure that all boxes are un-checked.
General\Type	Select **WCF-Custom** from the drop-down list and then click the **Configure** button, this will display the **WCF-Custom Transport Properties** dialog box which should be configured as described in the following section, **Configure the WCF-Custom Send Transport**.

Configure the WCF-Custom Send Transport

In the **WCF-Custom Transport Properties** dialog box displayed while creating the BizTalk Server Send Port, leave all properties at default values except as noted in the table below:

Property	Value
General\Address (URI)	net.tcp://<*Computer Name*>:2001/TCP1  **Important** <*Computer Name*> is a placeholder for the actual computer name used to host

Property	Value
	IndigoService.exe, which is designed to consume messages sent via WCF. Because IndigoService.exe requires very little resources, it is often perfectly acceptable to run IndigoService.exe on the SQL Server computer used for the BizTalk Server group databases. IndigoService.exe is part of the BizTalk Benchmark Wizard, which is available at BizTalk Benchmark Wizard (http://go.microsoft.com/fwlink/?LinkID=186347).
Binding\Binding Type	**customBinding**

As with most of the WCF-Custom Binding Types, the **customBinding** Binding type exposes several properties, which should be set to the following values:

1. Under the **Binding** section there is a **CustomBindingElement** property with an associated **Configuration** section. Leave all of the values in the Configuration section for the **CustomBindingElement** property at their default values.
2. Then under **CustomBindingElement** right-click **textMessageEncoding** and select **Remove extension (Del)**. Likewise, right-click **httpTransport** and select **Remove extension (Del)**.
3. Now right-click **CustomBindingElement** and select **Add extension Ins** to display the **Select Binding Element Extension** dialog box.
4. Select **binaryMessageEncoding** and click **OK** to add the **binaryMessageEncoding** element extension. Repeat the steps to display the **Select Binding Element Extension** dialog box and scroll down the list of available element extensions until you see the **tcpTransport** element extension, select **tcpTransport** and click **OK**.
5. Under **CustomBindingElement** select the **tcpTransport** element and in the Configuration section for **tcpTransport**, leave all properties at default values except as noted in the following table:

Property	Value
connectionBufferSize	2097152
maxBufferSize	2097152
maxPendingAccepts	400
maxPendingConnections	400
listenBacklog	400
maxBufferPoolSize	2097152
maxReceivedMessageSize	2097152

6. Under the **tcpTransport** element select the **ConnectionPoolSettings** element and leave all properties at default values except for the **maxOutboundConnectionsPerEndpoint** property, which should be changed to a value of 400.
7. Click **OK** to close the WCF-Custom Transport Properties dialog box, then click **OK** again to close the BTSLoadTest.Messaging.Send.WCF-Custom – Send Port Properties dialog box.

Configure a Computer to Consume Messages Sent by the BizTalk Server Send Port

As described earlier, the IndigoService.exe is designed to consume messages sent via WCF. IndigoService.exe is part of the BizTalk Benchmark Wizard, which is available at BizTalk Benchmark Wizard (http://go.microsoft.com/fwlink/?LinkID=186347). The easiest way to set up and run IndigoService.exe for purposes of this example is to simply download the BizTalk Benchmark Wizard zip file from the BizTalk BenchMark Wizard download page (http://go.microsoft.com/fwlink/?LinkID=209100) and extract the following 4 files onto the computer that you would like to run IndigoService.exe:

1. \IndigoService\bin\Release\IndigoService.exe
2. \IndigoService\bin\Release\IndigoService.exe.config
3. \IndigoService\bin\Release\Response.xml
4. \IndigoService\bin\Release\StartIndigoService.bat

Then, start IndigoService.exe by double-clicking on StartIndigoService.bat. IndigoService.exe consumes messages sent to the endpoint specified in the IndigoService.exe.config file:

```
<endpoint address="net.tcp://localhost:2001/TCP1"
          binding="netTcpBinding"
          bindingConfiguration="Binding1"
          name="endpoint1"
          contract="IndigoService.IServiceTwoWaysVoidNonTransactional" />
```

This is why the Send Port Address is configured with an Address (URI) of net.tcp://*<Computer Name>*:2001/TCP1

Because IndigoService.exe requires very little resources, it is often perfectly acceptable to run IndigoService.exe on the SQL Server computer used for the BizTalk Server databases.

Disable Tracking and Throttling for the BizTalk Server Group

In order to determine the absolute maximum sustainable throughput of the system, both message Tracking and Throttling should be disabled before beginning load testing. This can be done using the BizTalk Server Administration console by following these steps:

1. Launch the BizTalk Server Administration console. Click **Start**, point to **All Programs**, point to **BizTalk Server 2010** and then click **BizTalk Server Administration**.
2. Under **BizTalk Server Administration**, select your BizTalk Group if it is listed or if it is not listed, right-click **BizTalk Server Administration**, select **Connect to Existing Group**, enter the SQL Server name that houses the BizTalk Group's BizTalk Server Management database next to **SQL Server name:**, enter the name of the BizTalk Group's management database name next to **Database name:** and then click **OK**.
3. Right-click the BizTalk Group node and select **Settings** to display the **BizTalk Settings Dashboard**.
4. Click to select **Hosts** in the left hand pane of the BizTalk Settings Dashboard.
5. Click the dropdown list next to **Host** to select one of the hosts that will be used during performance testing.
6. Leave properties at their default values except as noted in the following table:

Property	Value
General\Move tracking data to DTA DB	Uncheck this box if it is checked.
General\32-bit only	Uncheck this box if it is checked.
General\Polling Intervals\Messaging	Set to a value of 20000000
General\Polling Intervals\Orchestrations	Set to a value of 20000000
Resource-Based Throttling\In-process messages	Set to a value of 10000
Resource-Based Throttling\Internal message queue size	Set to a value of 10000
Resource-Based Throttling\Message count in DB	Set to a value of 0
Resource-Based Throttling\Memory Usage\Process virtual	Set to a value of 0
Rate-Based Throttling\Publishing\Throttling override	Set to Do Not Throttle
Rate-Based Throttling\Delivery\Throttling override	Set to Do Not Throttle

7. Repeat the process outlined in step 6 for every host that will be used during the course of performance testing.
8. Click to select **Host Instances** in the left hand pane of the BizTalk Settings Dashboard.
9. Click the dropdown list next to **Host Instance:** to select one of the host instances that will be used for performance testing.
10. Leave property values at their default setting except change the **.NET CLR Maximum worker threads** to a value of **100** and change the **.NET CLR Minimum worker threads** to a value of **25**.
11. Repeat the process outlined in step 10 for every host instance that will be used during the course of performance testing.

Even though disabling Tracking and Throttling does not in any way represent what should be done in a production scenario, because these operations are so expensive from a performance perspective it makes sense to disable them to find out the true Maximum Sustainable Throughput (MST) of a BizTalk Server Environment. This allows testers to clearly see the impact of any performance tweaking that has been applied to the environment. Certainly an argument could be made that tracking should not be disabled and if you know from the outset that your BizTalk Server application will require tracking then you should enable tracking. With that said, **every effort should be made to disable throttling for purposes of performance testing**. Throttling is very useful for preventing a BizTalk Server from "falling over" due to excessive load in a production environment. However, you don't want throttling enabled during performance testing because it is expensive from a performance standpoint and because if throttling kicks in during a load test then you will have a very difficult time determining what level of performance your BizTalk Server application can actually achieve. The next topics describe how to perform step load testing to push your BizTalk Server environment beyond MST and then scale back to actual MST with constant load testing. If throttling is enabled then it will become nearly impossible to push your BizTalk environment beyond MST so that you could in turn discover what the true MST is.

Step 5: Perform Step Load Pattern Tests to Determine Maximum Sustainable Throughput

The simplest method for determining the Maximum Sustainable Throughput (MST) of a BizTalk Server solution with Visual Studio load testing is to perform a step load pattern and compare the total Documents received per second to the total Document processed per second. As long as the average total documents processed per second is greater than or equal to the average total documents received per second for the duration of the test, then the load is considered sustainable. If the average total documents received per second is greater than the average total documents processed per second for the duration of the test, then the load is not considered sustainable, and this will be evidenced by a corresponding growth in the value of the BizTalk:Message Box:General Counters\Spool Size counter. Over time, when a BizTalk Server application receives more documents than it can process, the unprocessed documents will accumulate in the MessageBox database, which will eventually induce a throttling condition and significantly degrade the performance of the BizTalk Server application.

Configure the Load Test with a Step Load Pattern Appropriate for your Application

Follow the steps in the topic Step 3: Create a Load Test to Perform Multiple Unit Tests Simultaneously to create a load test that uses a step load pattern. Factors that impact the ability of the BizTalk Server Application to process documents in a timely fashion include:

- **Number of BizTalk Server computers in your group** - Additional BizTalk Servers provide additional processing ability.
- **Size of the messages being processed** - Larger messages require additional processing resources.
- **Amount of document mapping performed** -Mapping requires additional processing resources.
- **Receive or send pipelines required by the application**. - Complex pipelines require additional processing resources.
- **Adapters and/or Accelerators used by the BizTalk Server application** – Some adapters and/or accelerators require more processing resources than others.
- **Amount of Message tracking required** – Message tracking is resource intensive.
- **Number of and complexity of Orchestrations running in the BizTalk Server Application** – Orchestrations can be very resource intensive.

When configuring the Step Load Pattern Test, modify the values specified for **Start user count** and **Maximum user count** to ensure that the number of messages specified for Start user count can be easily handled by the BizTalk Server application over time and likewise, the number of messages specified for Maximum user count is more than the BizTalk Server application can handle over time. See Add a Load Test and Configure the Load Test Scenario, Counter Sets, and Run Settings for information about editing the load pattern settings for the load test.

Ensure that the Correct Test Settings are Used for the Step Pattern Load Test

Configure the Load Test to use the **Test Settings** that you created in Add A Test Settings File to the Solution to Run Tests and Collect Data Remotely.

Configure the Load Test with the Appropriate Performance Counters and run the Step Pattern Load Test

Follow the steps in Add a Custom Counter Set to Measure BizTalk Server Key Performance Indicators (KPI) to add the necessary BizTalk Server performance counters which can be used to measure the performance of the BizTalk Server Application and determine at what point the BizTalk Server Application is no longer able to maintain the message load created by the Load Test Agents. This will be evidenced by the accrual of a backlog of messages in the Spool table as seen by an increased value for the BizTalk:Message Box:General Counters\Spool Size counter. If the value for this counter begins to increase significantly then you have likely exceeded the MST of your BizTalk Server Application. Once you have determined the number of messages at which the BizTalk Server Application is no longer able to process as many messages as it is receiving, make a note of the Documents received/Sec when this occurs. It is important to make a note of this value because the topic Step 6: Perform Constant Load Pattern Tests to Verify Maximum Sustainable Throughput will describe how to run a constant pattern load test with a “Constant User Count” value that is somewhat smaller than the maximum sustainable Documents received/Sec value. This is done to verify that the BizTalk Server Application is capable of processing this number of messages over time. To view values for counter sets, first start the

load test by right-clicking the test name (e.g. BTS_Messaging_Step) and then click the **Run Test** menu option. After Performance counters are initialized and the load test begins, Visual Studio will automatically switch focus to the Graphs window which allows you to display from 1 to 4 graphs simultaneously. If you are primarily interested in only viewing key performance indicators, as defined in Add a Custom Counter Set to Measure BizTalk Server Key Performance Indicators (KPI), click the **Panels** dropdown list from the Load test menu and select the option for **One Panel**. Then click the drop-down list at the top of the chart and select **Key Indicators** to display values for the key Performance indicators in real time.

Note

Because certain default counter values will be displayed in the **Key Indicators** graph and because you will probably want to display counter values that you added to your custom counter set, you may want to start by manually deleting each of the counters displayed in the **Key Indicators** graph and then manually add counters from your custom counter set(s). For example, at the least, you would want to add at least the counters in the table below to your graph to determine how well the BizTalk Server environment is handling the load, and where any bottlenecks may be occurring:

Counter Category	Counter	Instance	Computer
BizTalk:Message Box:General Counters	Spool Size	*BizTalk Server Message Box database*:*SQL Server Instance that houses the BizTalk Server Message Box database*	Any BizTalk Server in the group with the BizTalk Server Administration Console installed.
BizTalk:Messaging	Documents received/Sec	RxHost (or name of the receive host)	*BizTalk Server Computer#1 in the BizTalk Server Group*
BizTalk:Messaging	Documents received/Sec	RxHost (or name of the receive host)	*BizTalk Server Computer#2 in the BizTalk Server Group*
BizTalk:Messaging	Documents received/Sec	RxHost (or name of the receive host)	*BizTalk Server Computer#n in the BizTalk Server Group*
BizTalk:Messaging	Documents processed/Sec	TxHost (or name of the send host)	*BizTalk Server Computer#1 in the BizTalk Server Group*
BizTalk:Messaging	Documents processed/Sec	TxHost (or name of the send host)	*BizTalk Server Computer#2 in the*

Counter Category	Counter	Instance	Computer
			BizTalk Server Group
BizTalk:Messaging	Documents processed/Sec	TxHost (or name of the send host)	*BizTalk Server Computer#n in the BizTalk Server Group*
Processor	% Processor Time	_Total	*BizTalk Server Computer#1 in the BizTalk Server Group*
Processor	% Processor Time	_Total	*BizTalk Server Computer#2 in the BizTalk Server Group*
Processor	% Processor Time	_Total	*BizTalk Server Computer#n in the BizTalk Server Group*
Processor	% Processor Time	_Total	*SQL Server instance that houses the BizTalk Server databases*

Step 6: Perform Constant Load Pattern Tests to Verify Maximum Sustainable Throughput

When load testing a BizTalk Server solution using Visual Studio 2010, a constant load pattern test should be performed once the approximate Maximum Sustainable Throughput (MST) of the solution is determined as described in Step 5: Perform Step Load Pattern Tests to Determine Maximum Sustainable Throughput. This should be done to confirm that the MST is in fact sustainable over time and also so as to create a baseline load test moving forward to evaluate the effects of any performance tuning applied to the BizTalk Server application or environment.

Create and run a Constant Load Pattern Test

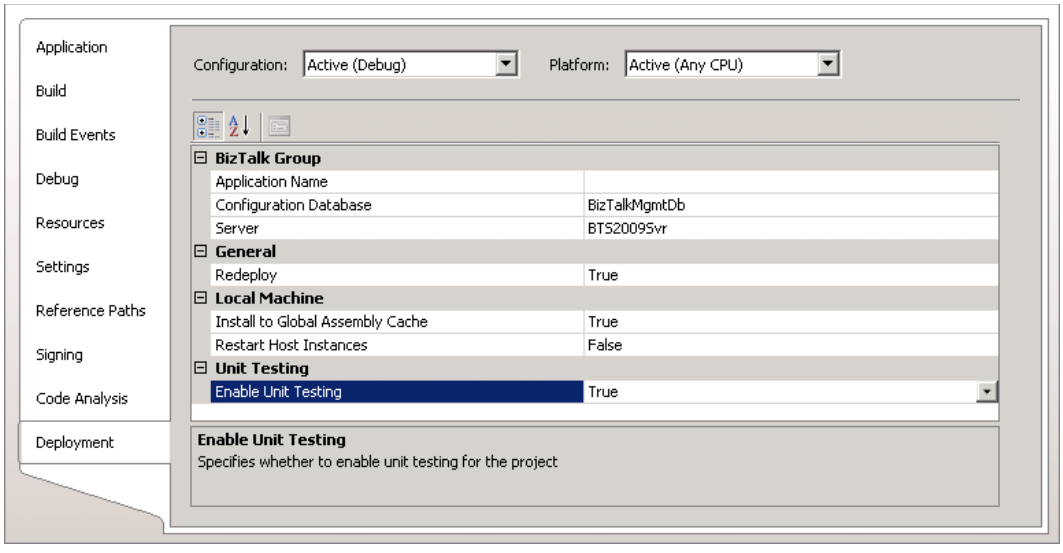
The easiest way to create a Constant Load Pattern test that uses the same Test Mix, Counter Sets, and Counter Set Mappings used by the Step Load Pattern test is to simply save the Step Load Pattern test “BTS_Messaging_Step.loadtest” as “BTS_Messaging_Constant.loadtest” and then make some changes to “BTS_Messaging_Constant.loadtest”. Follow these steps to create a Constant Load Pattern test that is based upon the existing Step Load Pattern test:

1. Open BTS_Messaging_Step.loadtest if it is not already open.
2. Click **File** and select **Save LoadTests\BTS_Messaging_Step.loadtest As**.
3. In the **Save File As** dialog box, next to **File name**, enter C:\Projects\LoadTest\BTSLoad\LoadTests\BTS_Messaging_Constant.loadtest and then click **Save**.

4. In the Load Test Editor, rename the scenario **BTS_Messaging_Step** to **BTS_Messaging_Constant**. The scenario name is displayed directly under the **Scenarios** folder.
5. Leave the values for **Test Mix** and **Network Mix** unchanged but click to select **Step Load Pattern**.
6. Right-click **Step Load Pattern** and select **Properties**.
7. In the **Properties** section, under **Load Pattern** click the dropdown list next to **Pattern** and change the Pattern from **Step** to **Constant**.
8. In the **Properties** section, under **Parameters**, change the value for **Constant User Count** to a value slightly less than the number of users at which the Step Load Pattern test was becoming unsustainable (i.e. the value for the **BizTalk:Messaging\Documents received per/Sec** began to consistently exceed the value for **BizTalk:Messaging\Documents processed/Sec** and the value of the **BizTalk:Message Box:General Counters\Spool Size** began to increase unbounded).
9. Under the **Run Settings** folder, right-click **Run Setting1 [Active]** and select **Properties**.
10. Scroll down the list of properties to the **Timing** section and enter a value for **Run Duration** of at least 10 minutes (00:10:00) and verify that the value for **Sample Rate** is still set to 5 seconds (00:00:05).
11. Start the load test by right-clicking the test name (e.g. BTS_Messaging_Constant) and then click the **Run Test** menu option.

Unit Testing

BizTalk Server 2010 introduces a unit testing feature that can be enabled on the **Deployment** property page of a BizTalk project. The following screenshot shows this project setting accessed from Project Designer when you right-click a project and click **Properties**.



Screenshot of the Deployment tab in Project Designer exposing the Enable Unit Testing project property

This feature allows you to create unit tests for schemas, maps, and pipelines. The topics in the BizTalk Server 2010 documentation provide some example approaches to using the unit testing feature. When this feature is enabled and the project rebuilt, the artifact classes will be derived from the following base classes to support unit testing.

Artifact type	Base class
Schema	**Microsoft.BizTalk.TestTools.Schema.TestableSchemaBase**
Map	**Microsoft.BizTalk.TestTools.Mapper.TestableMapBase**
Pipeline	**Microsoft.BizTalk.TestTools.Pipeline.TestablePipelineBase**

For more information about the unit testing feature introduced with BizTalk Server 2010, see the following topics in the BizTalk Server 2010 Help:

- Using the Unit Testing Feature with Schemas and Maps (http://go.microsoft.com/fwlink/?LinkId=150482).
- Using the Unit Testing Feature with Pipelines (http://go.microsoft.com/fwlink/?LinkId=150483)

See Also

Implementing Automated Testing