

Cool Features for SQL Server 2008

*written by
Geoff N. Hiten,
SQL Server Expert and
Microsoft MVP.*



© Copyright Quest® Software, Inc. 2008. All rights reserved.

This guide contains proprietary information, which is protected by copyright. The software described in this guide is furnished under a software license or nondisclosure agreement. This software may be used or copied only in accordance with the terms of the applicable agreement. No part of this guide may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's personal use without the written permission of Quest Software, Inc.

WARRANTY

The information contained in this document is subject to change without notice. Quest Software makes no warranty of any kind with respect to this information. QUEST SOFTWARE SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTY OF THE MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Quest Software shall not be liable for any direct, indirect, incidental, consequential, or other damage alleged in connection with the furnishing or use of this information.

TRADEMARKS

All trademarks and registered trademarks used in this guide are property of their respective owners.

World Headquarters
5 Polaris Way
Aliso Viejo, CA 92656
www.quest.com
e-mail: info@quest.com

Please refer to our Web site for regional and international office information.

Updated—January 2008

CONTENTS

- INTRODUCTION 1**
- ON-TIME RELEASE..... 2**
- OPERATIONS FEATURES..... 3**
 - POLICY-BASED MANAGEMENT 3
 - RESOURCE GOVERNOR 3
 - EXTENSIBLE KEY MANAGEMENT 4
 - MANAGEMENT DATA WAREHOUSE 5
 - DATABASE MIRRORING IMPROVEMENTS 6
- DEVELOPMENT FEATURES 7**
 - INTELLISENSE 7
 - ROW CONSTRUCTORS 8
 - MERGE 8
 - GEO-SPATIAL DATA TYPES 9
 - NEW DATE AND TIME DATA TYPES 10
 - TABLE-VALUED PARAMETERS..... 11
- CONNECT FEEDBACK 12**
- CONCLUSION 13**
- ABOUT THE AUTHOR 14**
- ABOUT QUEST SOFTWARE, INC. 15**
 - CONTACTING QUEST SOFTWARE..... 15
 - CONTACTING QUEST SUPPORT..... 15

INTRODUCTION

SQL Server 2008 is the latest version of this database platform from Microsoft, originally scheduled for release in the middle of 2008. Every SQL Server release brings a raft of new features, some highly sought after and some not so much. This paper will introduce a few features that will make a big impact on SQL administration and development.

ON-TIME RELEASE

Barring another Slammer-type episode, Microsoft will actually deliver SQL Server 2008 by the third quarter of 2008. This is a major departure from past releases when ship dates often slipped by over a year. In addition, we can expect the included feature set to be complete and working—no “maybe in Service Pack 1” features as we saw with database mirroring and SQL Server 2005. Microsoft is trying to get SQL Server on a 24- to 36-month cycle for major releases. This decision is driven by both technical and business considerations. Technically, such a release cycle drastically reduces the pressure to include features in service packs, simplifying the software maintenance process for both Microsoft and its customers. From a business standpoint, a five-year release cycle makes software assurance financially unattractive.

Unlike previous SQL Server releases from Microsoft, new features are not included in the main build line until they are almost complete. More importantly, no feature can build on top of another incomplete feature. Microsoft describes how this works in the beginning of almost every slide deck referencing SQL Server 2008. The most noticeable effect of this process is that each Community Technical Preview (CTP) has additional features instead of slightly more complete iterations of the existing features. This document is based on CTP5, released in November of 2007.

As a database administrator (DBA), you are either primarily operations-oriented or development-oriented. The features for SQL Server 2008 examined here are also divided into those two categories. These features are primarily aimed towards one category, but often do some work for the other one as well. This paper will start with the operations category and then move on to the development features.

One final note: some features may only be available in the Enterprise Edition of SQL Server. As of this writing, the feature mix is still in flux and will not be locked down until the actual product release. Check carefully before planning any design to make sure any necessary feature is available in the edition you use.

OPERATIONS FEATURES

Policy-based Management

Properly speaking, the Declarative Management Framework (DMF) is not a feature, it is a scenario. In Microsoft-language, a scenario is a group of individual features that offer a complete, interrelated set of functionality. There are several individual features that combine into the new DMF. The concept is quite simple and easy to describe. DBAs define policies such as the recovery model for read-only databases, guest log-in security, or auto-shrink for databases. They then point the policy to one or more servers and choose the policy type to implement. The goal of this scenario is to make managing a group of servers no more complex than managing a single server.

As with any new technology, there are new terms to learn. Microsoft breaks policies into targets, facets and conditions. Targets are the easiest term to understand. A target is any database object. These objects start at an instance and continue all the way down to a single database element such as an index or a view that can have a policy applied to it. Different policies may require different targets. As an example, a recovery model policy only would apply to database targets.

A facet is possibly the most difficult term to describe. A facet is a set of behaviors and characteristics for a particular target that can be controlled by a policy. Facets expose these characteristics as Boolean results called conditions. Put together, a facet set to a particular condition is determined by a policy. The real complexity of the entire DMF scenario is modeling behaviors and conditions into consistently controllable facets.

Policies can be run automatically or on demand. On-demand policies can be run either to check a server's current configuration or to set a server to match the policy. Policies can be grouped into categories for easier management. Automatic policies have three implementation options. One mode will prevent any changes from the set policy. A second mode will allow changes, but log out-of-compliance items. The third and final mode runs a scheduled check and logs any out-of-compliance items.

Resource Governor

SQL Server has always had one major critical weakness: a single, badly formed query can consume all system resources, rendering the system unresponsive for normal operations. Running an analysis-type query on a transaction processing system often generates major end-user complaints due to excessive CPU or memory consumption. SQL Server 2008 introduces a resource governor to limit the impact of a class of queries on a particular system.

Resource governor operates by dividing queries into workload groups. These can be classified by connection or user information. A classifier function divides queries at connection time into appropriate workload groups. Groups are assigned to resource pools. Each pool has different settings for memory and CPU. SQL resource governor cannot only limit the impact of one workgroup pool on an instance; however, it can guarantee a minimum allowance of memory and CPU for a specific group of queries. At installation, SQL creates a default resource pool and an internal resource pool. The internal pool is used for work that SQL server deems as essential to normal operation. As such, the internal pool settings cannot be altered, nor can the internal group classifier function be modified in any way. All other queries are assigned to the default group.

One of the simplest classifier functions is based on log-in name. Online transaction processing (OLTP) queries should use one log in, and analysis queries should use another log in to effectively use resource governor. The two sets of queries can be set to different resource pools, limiting the impact of a large aggregation query on a transaction processing system. The important concept to remember about resource governor is that it adjusts the balance between different resource pools. When all the active processes are in the same group, resource governor has no effect. While resource governor cannot completely solve the runaway query problem, it is a major step towards fixing a long-standing system weakness by allowing you to balance multiple workloads and reserve system capacity for specific connections.

Extensible Key Management

Many DBAs manage systems containing sensitive data that often come with specific regulatory compliance requirements. SQL Server 2005 introduced built-in encryption options to help protect sensitive data. Encryption adds an extra layer of protection to SQL Server data storage.

One weakness of SQL Server 2008 is that encryption keys are stored inside the same system as the encrypted data. This works well as a mechanism to protect data when it is not inside the server, such as for detached databases or database backups. However, it doesn't do as much good if someone has control of the entire system. Extensible key management allows SQL to use external key management modules called hardware security modules to encrypt data or other encryption keys.

Hardware Security Modules (HSM) should be familiar to system administrators working in core financial, securities or classified data centers. They provide several benefits over the built-in key management functions of SQL Server. HSMs are usually faster when encrypting or decrypting, and they require little or no host computer overhead. More importantly, they provide a mechanism to centrally manage security implementation for an organization. This is especially important from a regulatory compliance standpoint to demonstrate complete, centralized control over data disclosure and distribution. Until now, each database server was an island of security. Now, SQL Servers can be part of a complete security infrastructure.

Management Data Warehouse

At the 2005 PASS Community Summit, a Microsoft representative (who shall mercifully remain nameless) gave a presentation on a design called the Performance Data Warehouse (PDW). The PDW was supposed to aggregate performance data from several servers. Unlike the original SQL health and history tool, the performance history was gathered using the newly implemented dynamic management views. The PDW was supposed to ship as a free follow-on offering for SQL Server after SQL Server 2005 shipped. As with many good ideas, this one got shelved for mysterious and unknown reasons—most likely due to time, money, or personnel shortages.

Fortunately, SQL performance tracking has been given a second life as a regular, supported component of the SQL system. Microsoft leveraged some primary SQL components--specifically SQL Agent and SQL Server Integration Services (SSIS)--to implement a distributed database monitoring solution. One server is designated as the repository server where all data is uploaded. Two major components, a data collection element and a data uploading element, implement data gathering. The two processes are wrapped in separate SSIS packages but they can be coupled together if desired.

The system can be configured to run with local caching and periodic upload (asynchronous), or direct gather and immediate upload (synchronous). Servers that are "close" from a network perspective should use synchronous mode while asynchronous mode is best for "distant" servers. Each server in the network can be configured independently. SQL Server Management Studio serves both to configure data collection sets and to view collected data. One particularly well-designed feature has the system fire off an on-demand upload whenever a server's information is queried so the user does not see any stale data due to upload latencies.

As with any new scenario, Microsoft includes enough pieces to demonstrate basic usability. Pre-defined data collection sets for standard DBA monitoring elements are included and can be used out of the box. While the data is stored in relational tables, Microsoft does not recommend or support direct table access. Views and stored procedures are the supported data access methods. Components for create, read, update, and delete are all built-in. Microsoft also includes programming APIs for end users to build their own custom data collection into the platform.

Database Mirroring Improvements

Database mirroring is one of the best advances in SQL Server high availability (HA) since clustering was introduced. As with any new HA feature, there are specific procedures and technology requirements that have to be learned by the entire community. One of the major reasons that full support for mirroring was delayed until the first service pack release was so the problems and limitations could be learned by a few early adopters with direct Microsoft support before the feature was unleashed to everyone. Now that this technology is better known and more widely implemented, Microsoft has discovered some opportunities to improve the situation for everyone.

One area where SQL Server database mirroring needs improvement is in the data stream between the principal and mirror servers. This data stream can be quite large, especially relative to the bandwidth available between remote data centers. In some cases, this may eliminate mirroring as an option since there is simply not enough network capacity to support a high throughput system. Microsoft has added data compression to the log stream for database mirroring. Anytime compression is used, there is a CPU impact. However, the overall throughput of the system should be faster if network bandwidth is a constraining problem with a particular database mirroring application.

A second and possibly more useful feature is automatic broken page repair. A broken or torn page happens when there is either a hardware error at the storage layer or a page gets partially written to disk. With databases routinely getting over 100 GB in size and growing even larger, the statistical probability of storage failure also increases. Such a failure usually requires either a DBCC DBREPAIR that will possibly lose some data or requires reloading the entire database from a known good backup. Both procedures are a last resort type of operation that requires shutting users out of the system while it is being repaired. Mirroring offers a third, non-intrusive alternative. Provided some specific conditions are met, a mirrored pair can replace a broken page on the partner. The conditions differ depending on whether the page is broken on the principal or the mirror database so read BOL (Microsoft books online) carefully. Since this happens automatically, any errors that trigger page replacement do not get reported up to the client. This makes monitoring for storage subsystem errors more important than ever. Hardware errors are a lot like ants: if you see one, you know many more are following right behind.

DEVELOPMENT FEATURES

Intellisense

Sometimes it is the little things that make a lot of difference. SQL Server is the last major development environment from Microsoft to get Intellisense—Microsoft’s auto-complete technology. Admittedly, being last is neither unexpected nor unreasonable. The SQL environment is more challenging than other development platforms. The objects residing in external databases rather than in local code repositories create a much more complex name space in the SQL environment. Third-party vendors have tried to fill this gap with varying degrees of success. Initial reports are that the technology still has a lot of rough edges and sometimes slows down both client and server when populating the internal matching structures. But Microsoft still has improvement time allocated to this feature. Evidently, Intellisense was intentionally included earlier in the CTP release cycle than it might have been since Microsoft wants a lot of customer feedback on this feature.

Some of the limitations of Intellisense are obvious; some are more subtle. The obvious limitation is that a query has to be edited while connected to a database in order to resolve database-specific entities such as table, view, and procedure names. One of the more subtle issues is how Intellisense figures out what to recommend. The problem is scope-centric. Let’s take a SELECT statement as an example. Starting with “SELECT Col1, Col2, ...” and continuing on does not give good results with Intellisense. The suggestion box has a lot of options and most do not match what you want. Until a FROM clause exists, Intellisense begins by scoping names at the database level and expects you to write a multipart object name. It has no idea which table you are specifying columns for. However, if a FROM clause exists, the column names are populated correctly. Aliasing table names in a query allow Intellisense to work even better.

Building a query from the inside out reduces the scope that Intellisense has to search for candidate values. That in turn reduces the impact on the target server, speeds up the development system, and reduces developer frustration. As with any helper tool, Intellisense comes with an off switch if a particular developer’s style and Microsoft’s requirements clash too much.

Row Constructors

For several years, you have probably used a rather ugly kludge in order to treat a result set generated from a stored procedure like a normal result set. The code usually looks something like this:

```
Insert into #MyTempTable  
  
    Select * from OpenQuery(LoopBackServerName, 'exec MyStoredProc')
```

You then need to use the temporary table in an ordinary query. This requires you to create a loopback SQL Server by enabling the local server for data access. In SQL Server 2008, this ugly workaround is now unnecessary.

Row constructors allow you to directly load any table-formatted result into a permanent or temporary table.

```
Insert Into #MyTempTable  
  
    Exec MyStoredProc
```

The table above is now a valid SQL Construction. While this is a big improvement, it is still not a perfect solution. Row constructors are only available as part of the INSERT statement. "SELECT INTO" does not have this option so creating a table and inserting from a stored procedure is still not possible. The rest of the usual workarounds are still required, but loading a temporary table from a procedure just got a lot simpler and easier. It also got more secure since there is no loopback server requirement. As with any INSERT statement, the column data types and sizes must be implicitly convertible or SQL will generate an error.

Merge

SQL Server 2005 users first got a hint that Microsoft was going to implement an "upsert" command from the keyword highlight function of the Query window in SQL Server Management Studio (SSMS). Type "MERGE" into a query window in SSMS for SQL Server 2005 and it turns blue (**MERGE**), indicating a reserved keyword. Merge looks and works a lot like the CASE keyword, except it acts for an entire row rather than for a single column. MERGE also has a limited number of switch options with very specific actions.

Merge is structured similarly to an Insert or an Update statement, glued together with some control logic that strongly resembles a CASE statement. The MERGE statement specifies a:

- target table or alias to merge into
- table source to merge from
- search condition to define a match

- multiple optional action statements

The target table or alias is self-explanatory while the table source is a bit more complex. As with an insert or an update, the table source can be any table, real or derived, that results in a table-structured data set. Row constructors are not available in the MERGE statement. Any data from a stored procedure must be stored in a table, temp table, or table variable by one statement before it can be used in the MERGE statement.

The final elements of the query are separate statements to execute depending on the outcome of the matching condition. The MERGE statement is quite robust and allows multiple combinations of MATCHED, SOURCE NOT MATCHED, and TARGET NOT MATCHED sections. There are limitations on specific combinations of these sections and a serious warning on one particular use. You are likely to stumble over a limitation when writing complex MERGE statements. While a single row MERGE may insert some columns and update others, only one action may be performed on a row/column pair in a single MERGE statement.

The serious warning is for developers who compare two tables for completeness. If a MERGE statement uses both the SOURCE NOT MATCHED and TARGET NOT MATCHED sections, the two table sets are checked with a full outer join to execute the comparison. Any ON clauses in the comparison statements do not remove rows from evaluation. Such clauses merely change the outcome of the comparison, moving the row from the SOURCE NOT MATCHED code to the TARGET NOT MATCHED code or vice versa. The full tables (real or constructed) specified in the source and target queries are always compared in their entirety, potentially generating a very large intermediate result set. When importing even a small data set into a very large table, this can unintentionally result in an extremely resource-intensive query that can severely impact server performance.

Geo-Spatial Data Types

SQL Server specializes in processing transactions on structured data. Until SQL Server 2008, the only data elements available were represented using simple data types such as integer, floating-point, datetime, and character strings. SQL Server 2008 introduces a new complex data type into the SQL Server universe. It now understands, stores, and compares two new data types: geographic and geometric. Once you begin working with some data, the difference is simple and obvious. Geometric data is based on data points on a planar surface. It is based on an ellipsoid surface that conveniently uses the exact same latitude, longitude and distance metrics used to navigate and locate on the Earth's surface. The structure and use of geographic data conforms to widely used standards from the Open Geospatial Consortium (OGC). OGC represents geographic data in a SQL-compliant database. This will make interchanging data with new or pre-existing applications in text or binary format very easy.

Both data types can store multiple complex structures. Points, lines, polygons and collections of these base types can be stored in both the geometric and geographic data types. Simple SQL queries can describe characteristics of a single object or the relationship between two objects. Locating the center of an object or the shortest distance between two objects is now a single native SQL function. As with any scalar-valued function, the results can be used in WHERE and ON clauses, and

column outputs. Adding distance, intersection, overlap, or exclusion calculations to any query just became very simple.

One of the biggest challenges in dealing with any geospatial data type is scalability. Since databases routinely manage data sets of anywhere from hundreds to millions of rows, scalability is a serious concern. Comparing distances requires that every possible pair of data elements be compared and calculated. You need to create some type of indexing scheme to make such comparisons scalable. Microsoft solved this problem by taking the geo solution space and dividing it into sections. The shape of each section depends on whether geometric or geographic data is being indexed. Since geometric data is located on a theoretically infinite plane, a geometric index needs to have boundaries defined. Geographic data is always bounded by the physical dimensions of the Earth. Each geo-data element that has a component within each section is indexed for that section. The result is not a precise lookup as with traditional SQL indexes, but it does serve to reduce the possible solution space considerably. In theoretical programming terms, the “Big O notation” has been reduced from an N-squared to an N (Log N) solution. The practical result is a scalable geo-spatial solution built in to the database server.

New Date and Time Data Types

The original datetime data type has been a real workhorse over the years, but it has always had limitations that made it unsuitable for many implementations. Sometimes datetime is too small and doesn't reach far enough back in time. Other times it is too large and cannot record times with millisecond and sub-millisecond precision. Two new data types for Microsoft SQL Server – date and time - help to address these two issues. Date ranges from January 1st, 0001 through December 31, 9999. Most importantly, it stores only the dates without any time component. This makes it imminently suitable as a natural key for a conforming date dimension. While date does not handle Julian and Gregorian calendar conversions (or any other calendar conversions), it is now possible to record dates in a continuous data type from January 1, 0001 through December 31, 9999.

Besides holding only a time element without the corresponding date element, time as a data type, has one major improvement over using just the time portion of the datetime type. Time has a maximum precision of up to seven decimal places of fractional seconds, which is much higher than before. SQL Server can now store time with an accuracy of up to 100 nanoseconds. Should a particular database design not need quite that much detail, a lower precision can be selected to reduce the required storage space. The prior attempt to build these data types were dropped during the Beta release of SQL 2005. However this time, the implicit conversions to and from the existing datetime data type work intuitively.

It is hard to believe that after a long internal debate, Datetime2 is the name Microsoft chose for an extended datetime data type. But you are not stuck with this data type. It is harder to believe that “BigDatetime” did not make the cut. Datetime2 is simply a combination of the features of both the new date and time data types in one component. Datetime2 has the date range from the date type and the time precision of the time type.

The final new member of the date- and time-related data types is a definite plus for developers working on distributed applications with users in multiple time zones. Until now, you had to choose from one of several imperfect ways to handle multiple time zones. The simple ones such as forcing the entire company to run on “headquarters time” are usually unsuitable for a web presence and tend to irritate anyone not located in the “headquarters time” zone. The complex ones, such as tracking the time zone offset in a separate column, meant that native SQL datetime comparisons did not work. Since the data comparisons inside the database did not give a correct result, sooner or later someone in the development chain would miss the message and extract data without time zone adjustments. Shops with a lot of turnover were especially prone to this unique opportunity to provide wrong answers to business users.

Now there is a native solution that does not require users or developers to compromise usability or functionality. `Datetimeoffset` stores what the current datetime type does and it carries a time zone offset component as well. The offset is expressed in hh:mm format so that time zones such as Afghanistan (UTC +4:30) that run fractional hour offsets work exactly as expected. The comparison functions also work exactly as expected. The final treat in this feature is that it is based on the new, higher-precision `Datetime2` data type. Thus it carries the date and time range of the new data type.

Table-Valued Parameters

In order to overcome a fundamental SQL limitation, you used to create temporary tables to pass data from one stored procedure to another. This led to code that was difficult to trace and debug. T-SQL code became hard to performance tune since the temporary objects prevented SQL Server from generating estimated execution plans. Unintentional reuse of a name would lead to scope conflicts, unexpected side effects, and a generally unmaintainable T-SQL mess. The (obvious) solution is to use table variables and pass them as parameters. Now SQL Server allows such constructions.

Table-valued parameters are now legal in T-SQL programming. Unfortunately, there are some restrictive limitations on exactly what can be passed into a stored procedure. The primary restriction is that the table variable can only be passed as a `READONLY` parameter. Data can be passed into a calling chain, but it cannot be passed back up as a return value. This is not as limiting as it sounds, since the final `SELECT` statement of a procedure call stack is returned to the client as a result set anyway. You may have to adapt to methodologies to take advantage of this feature. But anyone who has worked with Microsoft products for any length of time should be used to adapting to the “Microsoft Way”.

Style adjustments may be well worth the trouble since table-valued parameters do not force nearly as much procedure recompilation as do temporary tables. It is also much easier to troubleshoot since there are no more “where does THAT come from?” moments when reading code with many temporary tables. If you are an operations DBA, you will be able to performance tune the entire stack without resorting to halfway re-writing the code to force temporary table creation.

CONNECT FEEDBACK

The final cool feature of SQL Server 2008 may not seem like a feature in the traditional sense, but it may be one of the more important changes in Microsoft SQL Server since its initial release. Once upon a time, Microsoft created the email address 'SQLWish@Microsoft.com'. This email address was the entire customer feedback mechanism for SQL Server prior to SQL Server 2005. During the Beta/CTP process for SQL Server 2005, Microsoft built the first iteration of a web-based, two-way feedback mechanism. "[Http://Connect.Microsoft.com](http://Connect.Microsoft.com)" is the current feedback system for SQL Server as well as other Microsoft products. Not only can end-users report bugs directly to Microsoft, but the SQL development team is accountable for acknowledging and resolving those bugs. Sometimes the answer is "By Design" or "Will Consider for Future Version", but at least there is some interaction with the development staff. A submitter can track bugs and provide repro code or impact statements. Other submitters can vote on the importance of an item. More votes translate directly to more visibility within the development team. This usually results in a higher priority and more resources. Submissions do not have to be just error reports; they can be suggestions as well. The new development methodology for SQL Server means that certain teams will have some slack time close to a release date while other code elements are finalized. This means there are resources available for fit-and-finish type items such as IU tweaks, additional tools, or fixing some performance issues in SQL Server Management Studio. Choosing between specific changes is heavily influenced by the Connect feedback and voting system.

CONCLUSION

SQL Server 2008 represents a major evolutionary step for Microsoft SQL Server. As with most core systems such as SQL, adoption is driven by one of two factors. Either there is a new project that uses the latest technology to fend off software obsolescence, or there is an existing system that needs one particular feature that is only available in the newest product release. Which feature is the compelling upgrade driver varies from system to system and shop to shop. Sometimes there is no feature and systems can coast on the current platform. Either way, the features descriptions here should help with your particular upgrade decision.

ABOUT THE AUTHOR

Geoff Hiten is a SQL Server consultant and Microsoft MVP based in Atlanta, Ga. He began working on SQL Server in 1992 with version 4.2. He specializes in highly available and high-performance SQL systems. Recent projects include system upgrades, platform migrations, performance tuning, custom reporting solutions, and database platform implementations. He is also involved with the Atlanta area Microsoft Database Forum Users group (AtlantaMDF). Hiten can be found lurking in the halls at PASS Community Summit events. He can be reached at SQLCraftsman@gmail.com.

ABOUT QUEST SOFTWARE, INC.

Quest Software, Inc. delivers innovative products that help organizations get more performance and productivity from their applications, databases and Windows infrastructure. Through a deep expertise in IT operations and a continued focus on what works best, Quest helps more than 50,000 customers worldwide meet higher expectations for enterprise IT. Quest Software can be found in offices around the globe and at www.quest.com.

Contacting Quest Software

Phone:	949.754.8000 (United States and Canada)
Email:	info@quest.com
Mail:	Quest Software, Inc. World Headquarters 5 Polaris Way Aliso Viejo, CA 92656 USA
Web site	www.quest.com

Please refer to our Web site for regional and international office information.

Contacting Quest Support

Quest Support is available to customers who have a trial version of a Quest product or who have purchased a commercial version and have a valid maintenance contract. Quest Support provides around the clock coverage with SupportLink, our web self-service. Visit SupportLink at <http://support.quest.com>

From SupportLink, you can do the following:

- Quickly find thousands of solutions (Knowledgebase articles/documents).
- Download patches and upgrades.
- Seek help from a Support engineer.
- Log and update your case, and check its status.

View the ***Global Support Guide*** for a detailed explanation of support programs, online services, contact information, and policy and procedures. The guide is available at: [http://support.quest.com/pdfs/Global Support Guide.pdf](http://support.quest.com/pdfs/Global%20Support%20Guide.pdf)