QTool Documentation

Version 2.5, January 2003

Robert M. Bruckner, Roland Braito
Vienna University of Technology
Institute of Software Technology & Interactive Systems
Favoritenstr. 9/188, A-1040 Vienna, Austria, Europe
bruckner@ifs.tuwien.ac.at

1 Overview

QTool is a utility, which facilitates the integration of message queuing services and data warehouse loading tools. QTool version 2.5 offers queue management and monitoring, message queue write functionality, and a scheduler used for continuous loading a data warehouse from a message queue. QTool currently offers support for Microsoft Message Queuing and was designed, but is not limited to, using Teradata's TPump utility.

2 Usage

The design goal of QTool is to offer the functionality needed to continuously load a data warehouse from a message queue. Since most data warehouses are rather complex entities, a decision has been made to limit each QTool instance to serve exactly one queue and one "Continuous Loading Job". Please note, that in our environment one "Continuous Loading Job" typically consists of at least two BTEQ scripts and two regular TPump jobs being called alternately. In order to load several tables from different queues, simply start one QTool instance per incoming queue. The typical way to use QTool is to specify the queue to be accessed, and then choose the desired functionality by selecting the appropriate tab.

When launched, QTool displays the *QStats* tab shown in Figure 1. At the upper left, there are fields for specifying the queue name. The buttons at the lower left are used to create, delete and empty the specified queue, as well as to count the messages in it. In addition, it is possible to send a special job-rotating message to the queue, which causes the scheduler to stop the current running instance of TPump and to start a new instance.

The QStats display is shown in the right pane. It consists of a graph showing the number of messages in the queue by time. This data can be optionally written into a file using a tab-separated format. In order to reduce the amount of data, a polling interval (milliseconds) can

be chosen. Since a graph offers only qualitative information, the display can be toggled between a graphical and a textual view of the data, offering precise quantitative information.

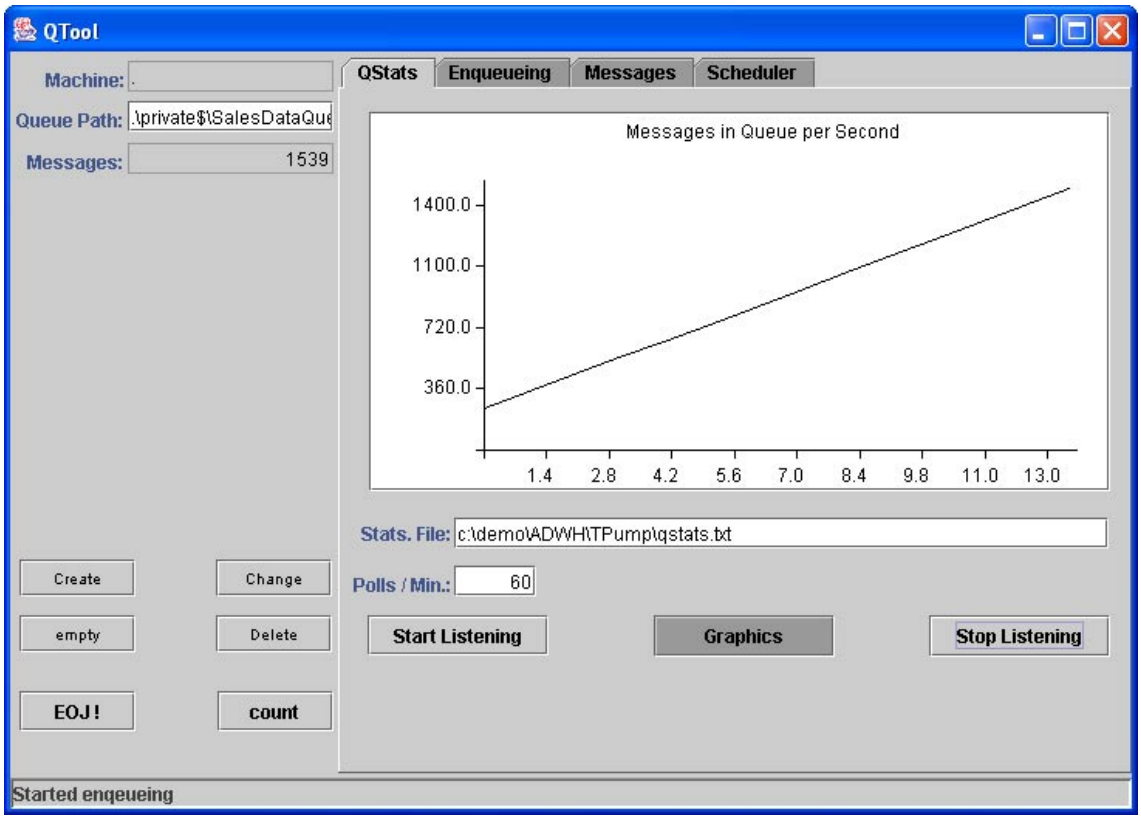

Figure 1: QTool QStats Tab

The QTool Queuing tab offers queuing functionality, which is one of the main purposes of QTool. The data snapshot is read from a flat file and written at the specified rate into the message queue. Additional properties can be specified optionally:

- message priority
- number of records per message
- maximum number of records to be queued

The lower part of the tab shows statistics and information, such as how long does it take to send one message, what is the maximum number of datasets from the flat file, which can be put into a single message, etc. These numbers are calculated during each calibration by inspecting the data-file. This calibration is sometimes necessary, because the time needed to compose and send messages depends on the message size and the available CPU power at the loading server. Calibration is initiated by clicking the ”Cal.” button.

We also implemented a skeleton of a messages tab for future use, which will provide access to messages in the specified message queue. It will list messages, offer the ability to inspect the properties and body of a message, as well as deleting a single message or several messages at once. By now, we use the Microsoft Management Console (see Figure 3), which

is part of the Windows 2000 and Windows XP operating system, for inspecting single messages.

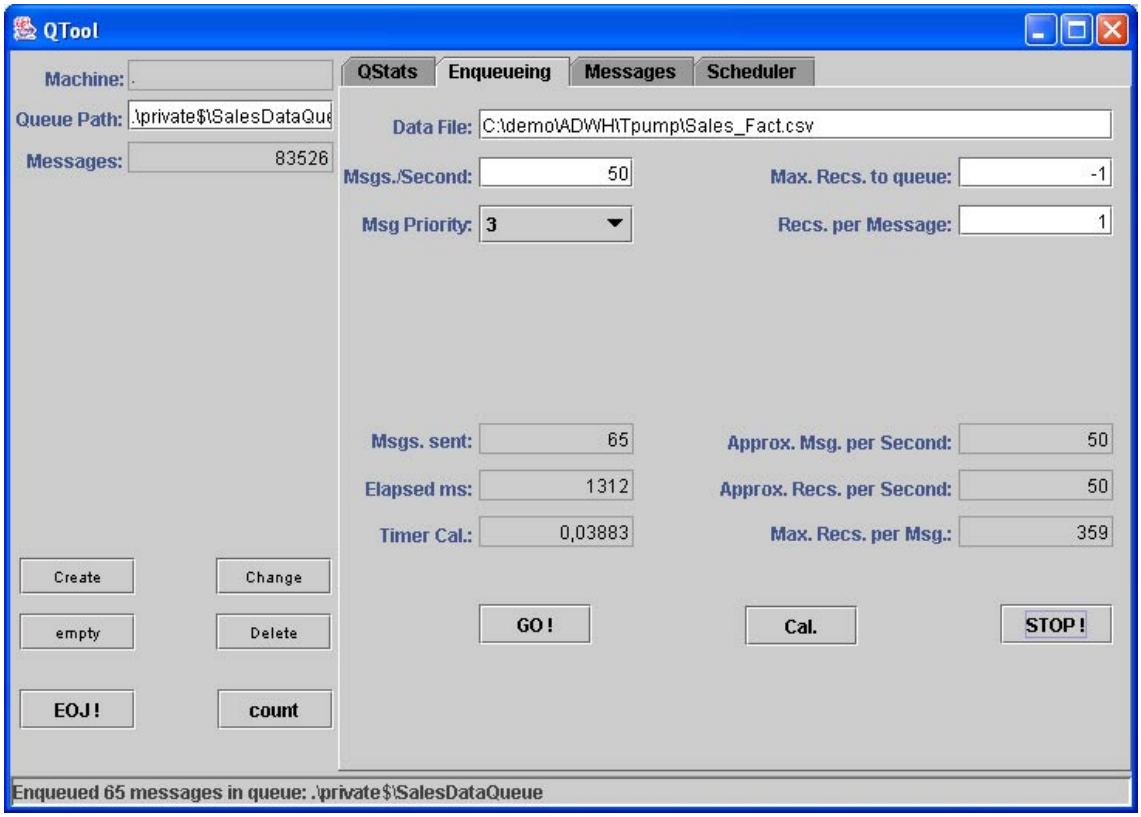

Figure 2: QTool Queuing Tab

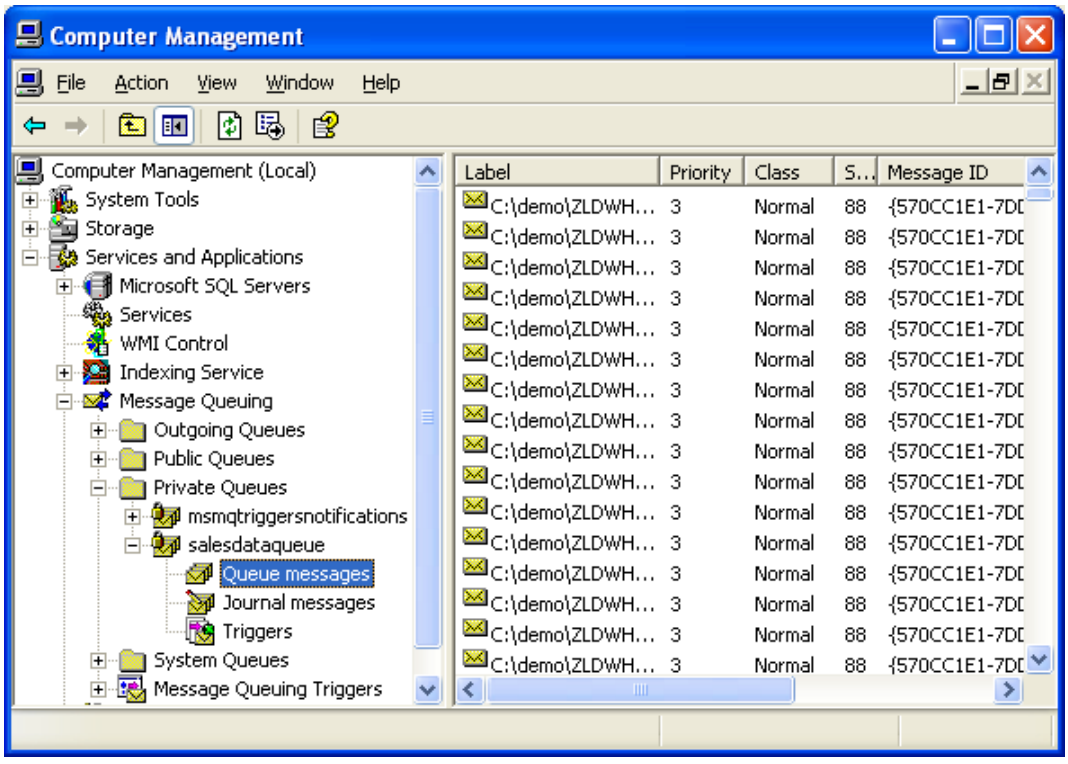

Figure 3: MSMQ Message Inspection

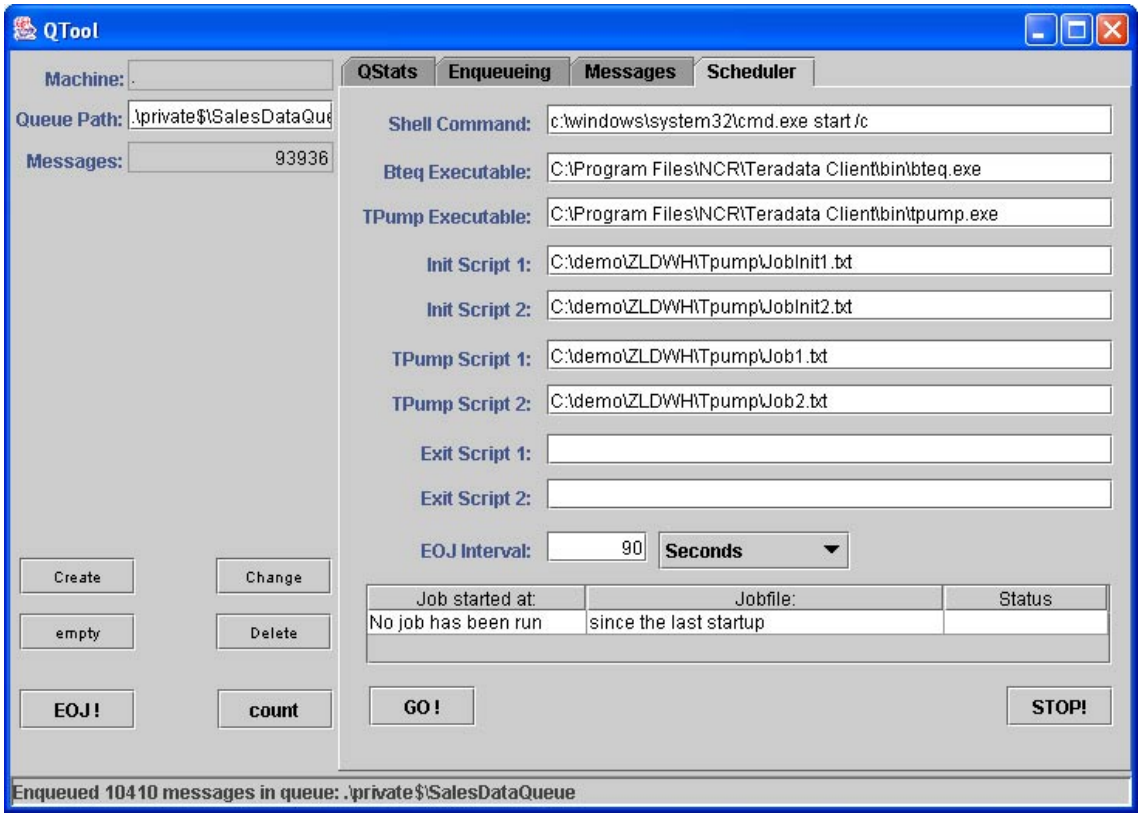

Figure 4: QTool Scheduler

The scheduler tab offers the ability to run a "Continuous Load Job". In our Teradata environment, a "Continuous Load Job" is comprised of two series of jobs each consisting of a BTEQ initialization job, followed by a TPump job feeding the data warehouse and a finalizing BTEQ job, which is intended to do some cleanup work in the data staging area. The loading jobs are alternately called to enable continuous data integration from a message queue into a data warehouse.

The QTool scheduler invokes external utilities and therefore needs to know where to find the executables and the job files. When starting the scheduler (by hitting "GO!"), it generates script files (batch files), which serialize the calls of the BTEQ initialization job, followed by the TPump call and the BTEQ cleanup job. The output is redirected into log-files. Every executed instance of a load job has its own associated log-file. The names of the log-files follow a specified scheme:

```
<nameOfJobFile>_out_<timestamp>.txt
```

`<timestamp>` has the format `YYYYMMDD_HHMMSS_mmm`. For example, the name of a log-file could be `LoadSalesFact_out_20021022_140222_223.txt`

The lower part of the scheduler tab provides a constantly updated table. It shows the history of started jobs as well as the current state of the active ones. Double-clicking a table-entry displays the corresponding output-file, thus simplifying log-file-checking and error tracking. Pressing the "STOP!" button stops the creation of new jobs, but does *not* interrupt currently running jobs, in order to make restarts of continuous loading jobs easier.

3 Implementation

QTool is written in Java and runs at J2SE and J2EE platforms. It uses Swing to display its GUI (and the JSci package for displaying the graph). Since the Microsoft Message Queuing Service (MSMQ) is neither Java Messaging Service (JMS) compliant nor provides Java-Interfaces, the Java Native Interface (JNI) has been utilized to access the MSMQ COM/C++ API and the Win32 API. This usage of native code is the reason of QTool's superior performance compared to a Visual Basic based tool, which was implemented at NCR's Active Data Warehouse Center of Expertise [Hahn, Ballinger 2001]; see section 5.

The *official* MSMQ API offers very limited management functionality. Therefore, QTool makes use of the *MSMQ Local Admin* API, which is not part of the official MSMQ SDK (software development toolkit) but is also used by some other Microsoft products internally. The MSMQ Local Admin API is accessed through a COM/C interface. Since Java offers a non-proprietary standard to use message services (JMS), we plan to add the capability to access JMS compliant messaging services.

Socket communication is used in order to enable high-speed communication between the native code accessing MSMQ and the Java code displaying *QStats* (part of the QTool functionality). QTool makes rather heavy use of multi-threading. The QTool scheduler uses the *Runtime.exec()* function to start the BTEQ and TPump jobs.

We plan to integrate JMS (Java Message Service) functionality that enables us to communicate with message-based applications in a (somewhat proprietary) standardized way. The interesting point about JMS is that many important middleware and messaging products vendors have licensed JMS and support the JMS API for accessing middleware functionality. Vendors include BEA Systems, Hewlett-Packard, IBM, Sun, Oracle, TIBCO Software and some others.

4 Installation

Compiling the QTool Java source files derives two components:

- **QTool_2_5.jar**
  This is a Java archive file for QTool version 2.5. It contains the QTool frontend functionality.

- **QToolNative.dll**
  This is a dynamic link library, which contains the QTool backend functionality. This includes MSMQ read, write, and management functionality (through JNI, COM/C, COM/C++).

  The dynamic link library should be copied to the system directory of the load server (e.g. Win2K: `c:\winnt\system32\`; WinXP: `c:\windows\system32\`).

QTool makes use of the Java Runtime Environment 1.3.0 or higher. The runtime is freely available and can be downloaded at http://www.sun.com/java. In order to start QTool from the command line enter: java –jar QTool_2_4.jar

5 Comparison and Discussion

Since QTool uses C/C++ code to access MSMQ, it is considerably faster than any Visual Basic (VB) tool. Our experiments with the Visual Basic-based data feeder tool [Hahn, Ballinger 2001] and the QTool revealed considerable performance differences. The data file in our experiment consists of about 41000 records (3.6 MB filesize). The results shown in Table 1 are based on various platform scenarios (described in [Bruckner 2002]). Additional experiments on other platform scenarios confirmed the observation that QTool is 3 to 4 times faster than the VB-based data feeder. When loading with the maximum data rate, both tools cause a very high CPU load. However, they are able to do loads with a specified (lower) data rate. This decreases the CPU load on the client and on the message queuing server.

Tool	Data Rate (Scenario 3A)	Ratio
VB-based data feeder	11280 messages/sec.	1
QTool data feeder	43390 messages/sec	3.85

Table 1: Comparison of Data Feed Performance

Although a single TPump job cannot cope with these very high data rates, high-speed data feeds are still important. Our approach on near real-time data integration with Teradata can scale-up by using several TPump jobs in parallel and by working on systems with a higher number of AMPs available in the MPP configuration.

6 References

[Bruckner 2002] Robert M. Bruckner:
Zero-Latency Data Warehousing – Toward an Integrated Environment for minimized Latency in Decision-Making.
Ph.D. thesis, Vienna University of Technology, November 2002.

[Hahn, Ballinger 2001] Robert Hahn, Carrie Ballinger:
TPump in a Continuous Environment.
Technical Report, NCR Active Data Warehouse Center of Expertise, April 2001.