QTool
An Overview

Roland Braito*

June 12, 2002

Abstract

QTool is small program which is being built to facilitate the integration of message queueing services and data warehouse loading tools. As of this writing, it offers basic queue management (including monitoring), enqueueing functionality, and a special scheduler used for continous loading a data warehouse from a message queue. QTool currently offers support for MS Message Queueing and was designed to be used with, but is not limited to, Teradata's TPump utility.

Contents

1 Overview

The principle design goal of QTool is to offer everything needed to continously 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 serving exactly one queue and one "Continous TPump Job". Please note, that one "Continous TPump Job" typically consists of at least two BTEQ and two regular TPump jobs being called alternately.

To fill several tables from different queues, simply start one QTool instance per queue to be accessed.

The typical way to use QTool would be to start it up, specify the queue to be accessed, and then choose the desired functionality by selecting the

*roland.braito@aon.at

appropriate tab. When launched, QTool displays the QStats tab which is shown below.

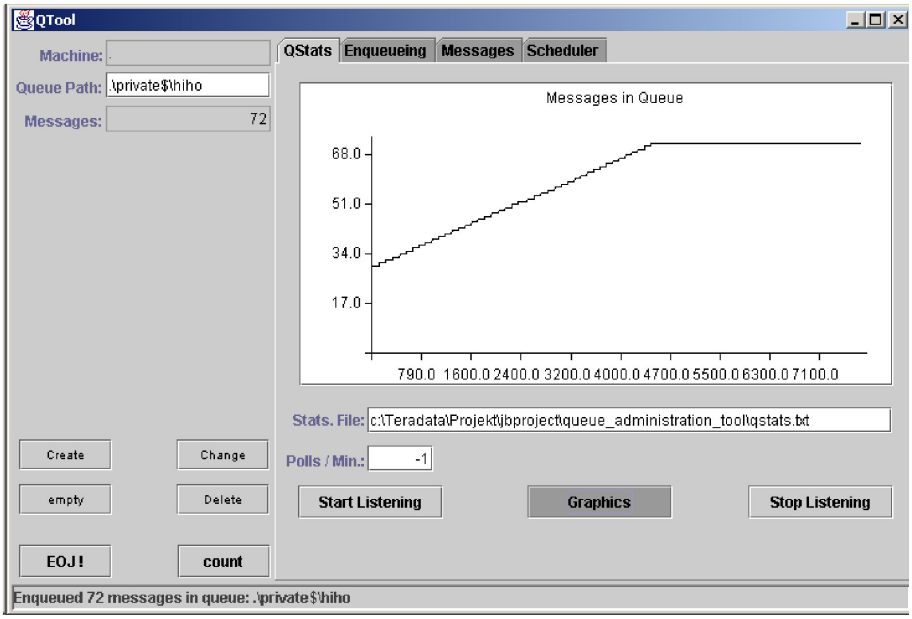

Figure 1: QStats tab showing graphical representation of the data

At the upper left, the fields to specify the queue are shown, 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 and to send a special job-rotating message to the queue. At the right, the QStats display is shown. It consists of a graph showing the number of messages in the queue plotted over time. These data are saved in tab-separated format to a file, which can be specified. To reduce the amount of data, a polling interval can be chosen.

At the bottom a status bar is shown.
Furthermore, since a graph offers only qualitative information, the display can be toggled between a graphical view and a textual view of the data, offering precise quantitative information.

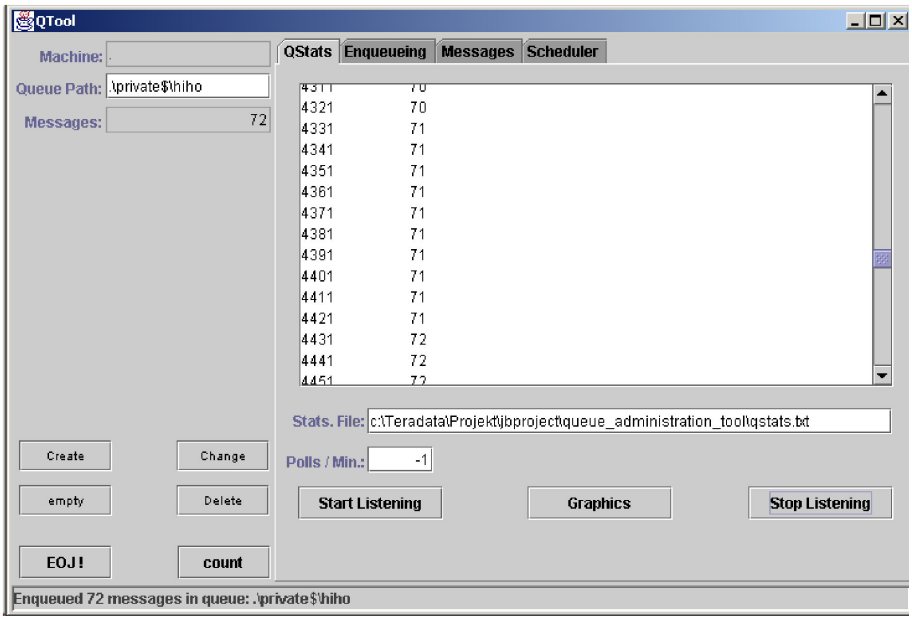

Figure 2: QStats tab showing textual representation of the data

The next tab offers enqueueing functionality, which is one of the main purposes of QTool. The data is read from a "flat file" and enqueued at the specified rate in the accessed queue. Further properties which can be specified are: message priority, how many records each message should contain and the total number of records to be queued.

In the lower part of the tab, some statistic are shown, along with how long it takes to send one message and how many records can be put into one message. The latter two are calculated during each calibration by inspecting the data-file. Calibration sometimes is necessary, due to the fact that the time needed to compose and send messages greatly depends on message size and CPU power. It is performed by klicking on the "Cal." button. Since QTool uses C++ code to access MS Message Queueing Service, it is considerably faster than any VisualBasic tool.

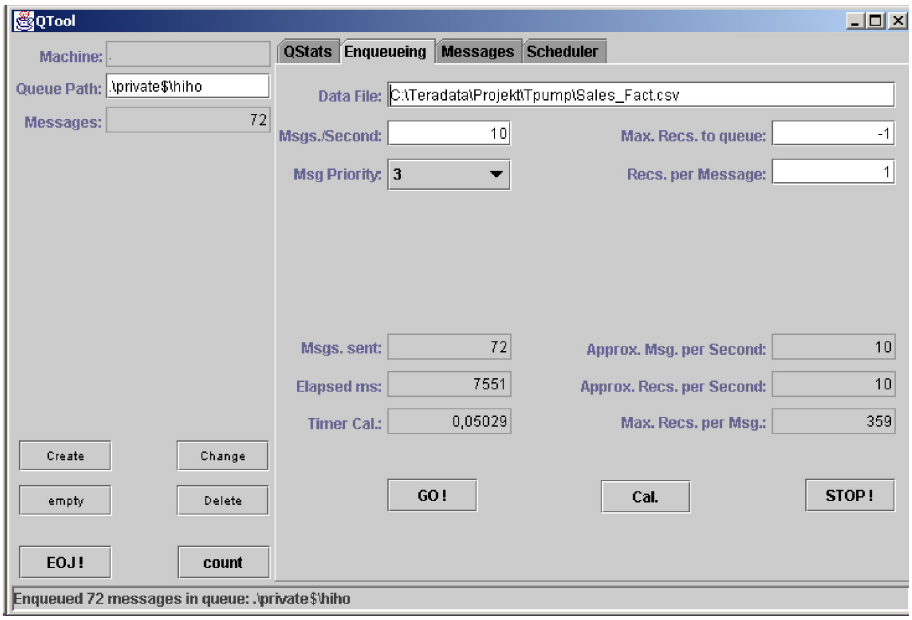

Figure 3: Enqueueing tab

The messages tab, which is not yet implemented, will provide access to messages in the specified queue. It will list the messages in the queue, offer the ability to inspect the properties and body of a message, as well as to delete a single message or several messages at once.

Until this functionality is implemented, the MS Management Console, which is part of the Windows 2000 operating system, has to be used to inspect single messages.

The scheduler tab offers the ability to run a "Continous TPump Job"[1]. A "Continous TPump Job" ist comprised of two series of jobs each consisting of a BTEQ initialization job, followed by a TPump job feeding the data warehouse and – optionally – a finalizing BTEQ job, which are alternately called to enable a virtually continous loading of the data warehouse. Since this is done through batch-processing, QTool needs to know where to find the executables and the job files. Once this information, along with the rotation interval, has been entered, and the scheduler has been started, QTool generates batch-files to call the BTEQ and TPump utilities, redirecting their output into files named according to the following scheme:

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

[1]For an in-depth discussion of this topic please refer to the article "TPump in a Continuous Environment" by Bob Hahn & Carrie Ballinger, NCR Active Data Warehouse Center of Expertise, April, 2001.

where

```
<timestamp>
```

has the format:

```
YYYYMMDD_HHMMSS_mmm
```

with hours being in 24-hour format and mmm being milliseconds. So, a possible output-file for the job-file

```
"D:\init1.txt"
```

would be

```
"D:\init1_out_20020522_140222_223.txt"
```

In the lower part of the tab, a constantly updated table is provided, showing the history of started jobs as well as the current status of the active ones. As a special bonus to illfated admins, a doubleclick on a table-entry displays the corresponding output-file, thus making error-tracking a little less painful...

Pressing the "STOP!" button stops the creation of new jobs, but does *not* interrupt already running jobs, since this can have a hazardous impact on the RDBMS.

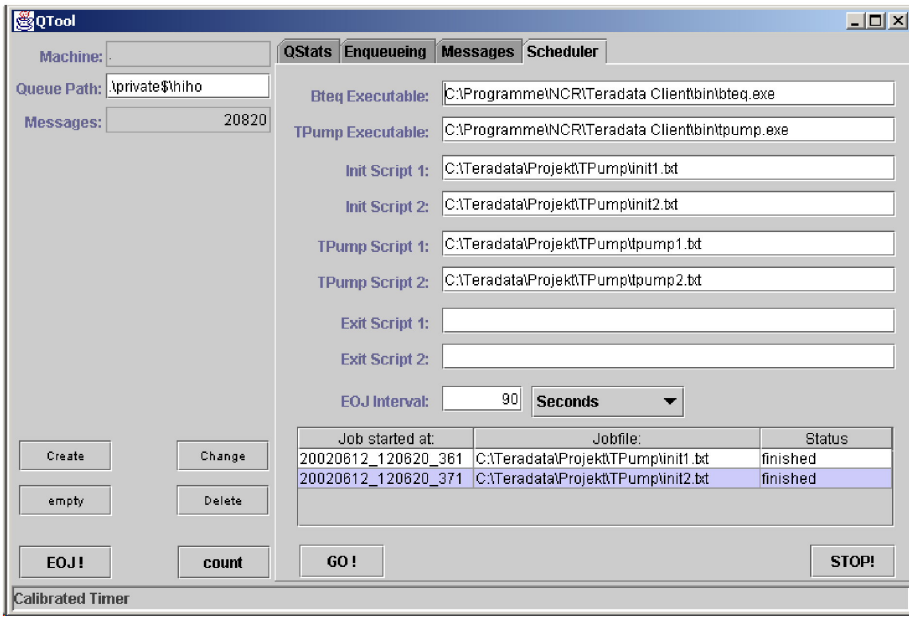

Figure 4: Scheduler tab

2 Future Work

Plans for the immediate future of QTool include:

- Implementing message reading and deletion
- Implementing further queue-management functionality (security, etc. . .)
- Implement JMS capability (see section 3)

3 Technology Used

> *Warning:* This section is aimed at readers interested in the technology behind QTool, and as such may cause considerable boredom to non-programmers!

QTool has been written in Java2SE. It uses Swing to display it's GUI (and the JSci package for displaying the graph).

Since MS Message Queueing 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 over VisualBasic based tools.

Unfortunately the standard MSMQ API is offers very limited management functionality. Therefore QTool makes use of the unofficial MSMQ Admin API.

Since Java offers a non-proprietary standard to use message services (JMS), it is planned to add the capability to access JMS compliant messaging services to QTool.

To enable high-speed communication between the native code accessing MSMQ and the Java code displaying the QStats, socket communication is being used.

The scheduler starts the BTEQ and TPump jobs using Runtime.exec(). Obviously, QTool makes rather heavy use of multithreading.