MQSeries Enabled Tcl Application

Ping Tong, Senior Consultant at Intelliclaim Inc., ptong@intelliclaim.com
Daniel Lyakovetsky, CIO at Intelliclaim Inc., dlyakove@intelliclaim.com
Sergey Polyakov, VP Development at Intelliclaim Inc., spolyakov@intelliclaim.com

As information technology is getting mature rapidly, the demand for new functionalities and the demand for preservation of complex legacy business logics have put the integration at the center of IT development in many companies. In this trend, message oriented middleware such as MQSeries, which serves as a data bus between different applications and different platforms, has played a critical role in gluing discrete components into a logically structured enterprise application system.

Since its birth about twenty years ago, Tcl has the vision to be extensible so that it could support new technologies as naturally as the basic language features. Tcl's extensibility along with its portability has fueled Tcl's popularity as a scripting language for integration in recent years. While MQSeries is becoming a dominant message-oriented-middleware, the need to interface with MQSeries from Tcl scripts has arisen. This article talks about a Tcl extension library "MQTCL" that serves as a bridge between Tcl application and MQSeries middleware, its internal architecture and design. In addition, this article explores the benefit of using MQTCL in enterprise integration.

The Role of MQTCL in Enterprise Integration

While MQSeries enables applications to communicate seamlessly in heterogeneous environment, Tcl provides an excellent programming tool for gluing various applications or components into a higher-level business application or a workflow. With MQTCL, a Tcl application can easily access MQSeries message and queuing infrastructure through simple Tcl commands. It can also engage itself into critical business transaction process through MQSeries transaction facility.

As shown in Figure 1, MQTCL enables a Tcl application to

- Communicate with applications written in various different programming languages such as C/C++, JAVA or COBOL
- Communicate with applications run on various platforms such as Windows, UNIX and Mainframe
- Communicate with various network protocols such as TCP, UDP, NETBIOS, LU62.

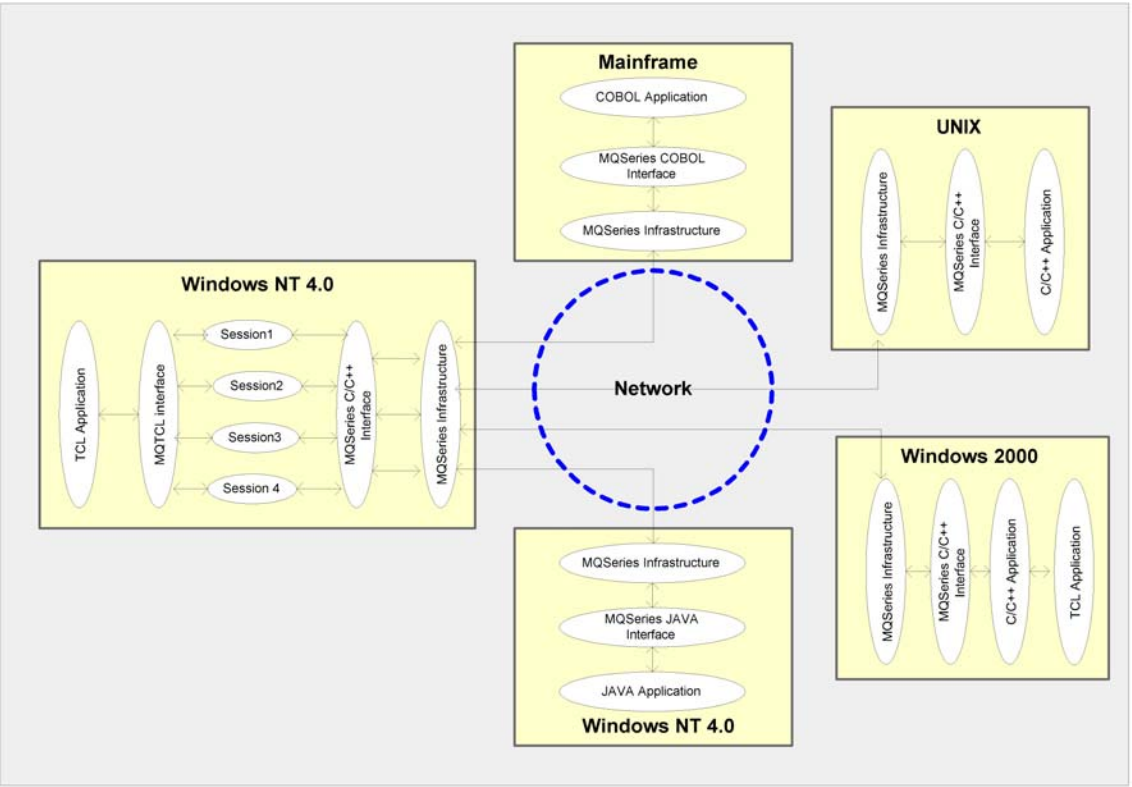

Figure 1: MQSeries enabled TCL application in a heterogamous enterprise environment

An important advantage of using MQTCL is that a Tcl can communicate with any kind of MQSeries applications locally or remotely without any internal knowledge of the applications. MQSeries shields various applications, system and network differences away from Tcl scripts. Combined with MQSeries cross-platform capability, Tcl becomes even more powerful in multiple platform integration.

Architecture of MQTCL Extension Library
MQTCL has two major modules internally as shown in Figure 2: MQTCL Wrapper and QAdapter. Most of MQTCL's functionalities reside in QAdapter module. MQTCL Wrapper simply converts features in QAdapter to TCL commands and exports them to TCL applications.

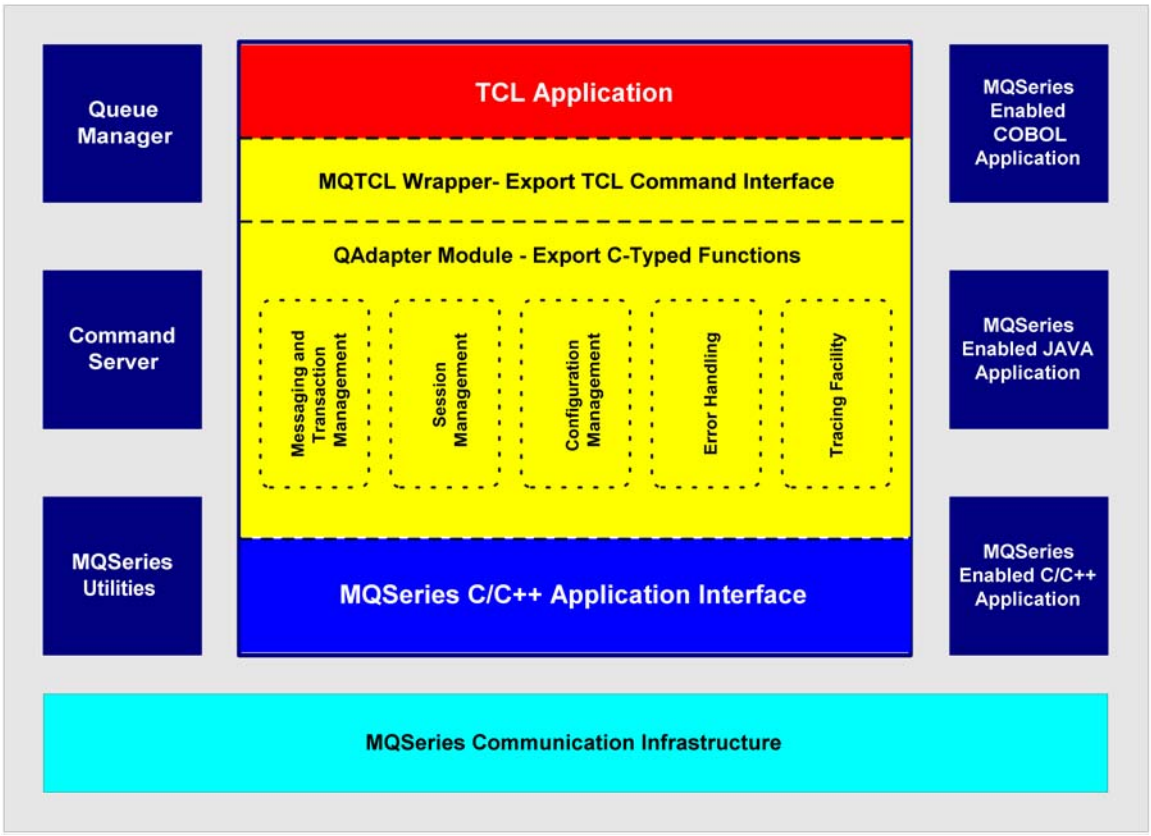

Figure 2: MQTCL Architecture Overview

MQTCL functionalities are grouped into five categories and are implemented as C++ class objects in QAdapter. The five categories are messaging and transaction management, session management, configuration management, tracing facility and error handling. Features in each category are exported as C-typed functions so that programs written in different languages can interface with QAdapter easily. QAdapter calls into MQSeries library and shields Tcl applications from the details of MQSeries system. In addition, portability has been an important consideration throughout the development of QAdapter. All system functions used by QAdapter are carefully selected to be POSIX functions.

In addition to the capabilities of sending and receiving MQSeries messages through simple Tcl commands, MQTCL supplies a rich set of other features. Below we will discuss each category of functionalities, and how to use Tcl commands to access these functionalities.

Session Management
To simplify the MQSeries programming in Tcl, MQTCL introduces session concept. Each session has a set of properties for MQSeries objects and programming options. Once a session is fully specified, Tcl commands can be issued to access MQSeries objects with minimum knowledge of underneath MQSeries system.

Each MQTCL session consists of an Input Queue, an Output Queue and a Backout Queue. The Input Queue and Output Queue are used to get and put messages. The Backout Queue is used to store messages that have been backed out for more than a specified limit, which is defined as a session property. MQTCL handles those backout messages automatically for you, and effectively avoids infinite loop in your Tcl applications.

MQTCL properties are divided into six sections:

- [QObjects] -- defines MQSeries objects for the session
- [GetMsgMQMD] -- specifies MQMD structure for getting messages off the input queue
- [PutMsgMQMD] – specifies MQMD structure for putting messages to the output queue
- [GetMsgOptions] – defines MQGMO structure
- [PutMsgOptions] – defines MQPMO structure
- [Logging] – specifies logging and tracing options

Three queues mentioned above with their associated queue manager must be defined in [QObjects] section. All the rest of properties are optional. If an optional property is not defined, a default value will be given by MQTCL according to MQSeries convention or MQTCL internally pre-defined values.

You can load session properties from a file, either a default file or a user defined file, through command interface or an environment variable. This nature allows users to configure their application environments without any Tcl code changes, and makes it easier when moving Tcl applications from QA environment to production environment. Example 1 demonstrates how to initialize a session from a user-defined file. You can also assign or change session properties at runtime through Tcl commands.

```
D:\>tclsh
% package require mqtcl
0.1.0.0
% mqtcl init -file d:\\mqtcl-test\\session1.cfg
1
% mqtcl info 1 -all
   session id     1
   qmgr:          MQTCL_TEST
   inputq:        QL.MQTCL.TEST.INPUT
   replyq:        QL.MQTCL.TEST.INPUT
   backq:         QL.MQTCL.TEST.BACKQ
   session_type:  0
   buffersize:    100000
   successlog:    d:\\mqtcl-test\\success.log
   errorlog:      d:\\mqtcl-test\\error.log
   successlevel:  0
   errorlevel:    2
   trace:         false
   <.....>
% mqtcl close 1
Ok
% exit
D:\>
```

Example 1: MQTCL Tcl commands for session initialization, session close and session information query.

If you need to interact with more than one input queue or more than one output queue, you simply initialize multiple sessions.

Internally an MQTCL session is a logical collection of objects that maintain the state of message exchanges. Each initialized session has an associated session ID. You use the session ID to access session properties and MQSeries objects. The lifetime of a session starts with MQTCL "init" command, and ends with MQTCL "close". A MQTCL application must initialize a session before it can issue any other MQTCL commands.

Messaging and Queuing

MQTCL exports MQSeries interfaces to Tcl application as Tcl commands. A Tcl applications loaded with MQTCL can communicate with other MQSeries applications written by different languages as well as on different systems because MQSeries provides programming interfaces for application in various languages for over 35 operating systems.

With MQTCL, generating MQSeries messages or receiving MQSeries messages are as easy as issuing any standard Tcl commands. Example 2 shows how to use MQTCL commands to put a message to a queue and

get a message from a queue, and how to display the message in a text format. A session ID, which is returned from session initialization, is required in MQTCL's put/get commands.

```
D:\>tclsh
% package require mqtcl
0.1.0.0
% set session1_id [mqtcl init -file d:\\mqtcl-test\\session1.cfg]
% mqtcl get $session1_id
% mqtcl display
"Hello MQTCL!"
% mqtcl put $session1_id "Reply Message: Hello client1."
% mqtcl close $session1_id
% exit
D:\>
```

Example 2: MQTCL Tcl commands for receiving, sending and displaying messages

On Windows, you can also verify the message you have put in Example 2 through MQSeries Explorer utility tool. Figure 3 is the screen capture of MQSeries Explorer dialog. The selected item in the dialog displays that a message was put into "QL_MQTCL_TEST.REPLY" queue by user "PTong" and application "D:\Tcl\bin\tclsh.exe" at 10/16/01 11:50:32AM.

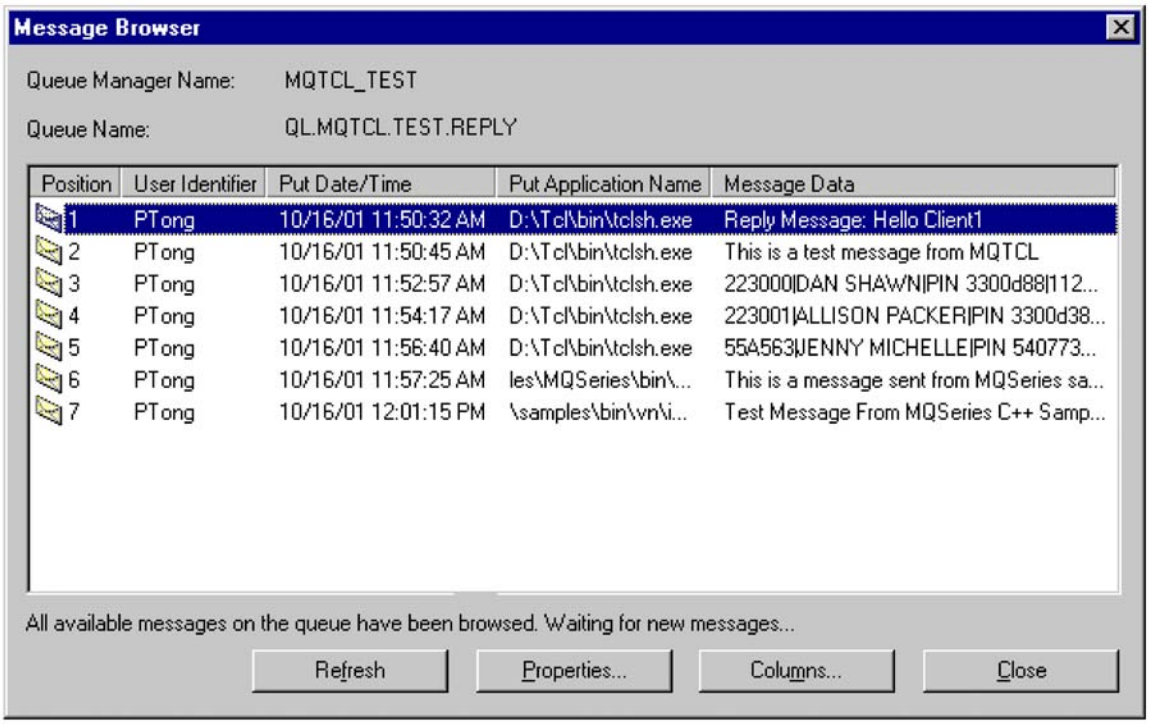

Figure 3: MQSeries Explore for browsing messages in a local queue

Transaction Support

Transaction support requires a set of actions to happen as a single unit of work in order to ensure data integrity. MQTCL application provides Tcl commands that are mapped to MQSeries transaction facility.

MQTCL "commit" command can be used to commit a unit of work. At this point all message exchanges made within that unit of work are made permanent and irreversible. Alternatively, all message exchanges can be backed out if the unit of work fails and MQTCL "rollback" command is issued.

When MQTCL application gets a message from a queue within a unit of work, MQSeries will mark the message unavailable. If the application commits the unit of work, message is then permanently deleted from the queue. If the application rolls back the unit of work, MQSeries restores the message in the queue.

Error Handling

Errors, such as unexpected hardware problems or unanticipated actions by the user, can occur no matter how carefully the code is written. Without a proper error handling, those run-time errors can halt execution

and make users unable to resume the application. MQTCL provides an error handling mechanism for handling errors occurred internally in MQTCL and in underneath MQSeries.

When errors occur, such as MQSeries facility is not available, MQTCL raises them to the upper Tcl calling procedure. Tcl application can catch these raised errors at run time, evaluate them and handle them accordingly. MQTCL also provides a Tcl command for accessing detailed error information such as error code id and error description in text format. Example 3 shows how to use MQTCL error handling to catch errors in MQTCL get call.

```
D:\>tclsh
% package require mqtcl
% set session1_id [mqtcl init -file d:\\mqtcl-test\\session1.cfg]
% if { [catch { mqtcl get $session1_id } ] } {
      if { [mqtcl errorcode -id] == 2033 } {
                  # no message in the queue, consider it as normal and continue
                  continue
      } else {
                  error "[mqtcl errorcode -string]"
      }
}
Error Code: 2016, Error: MQGET failed with reason code 2016
% mqtcl close $session1_id
% exit
```

Example 3: MQTCL Tcl command for error handling

In this example, application was coded to resume if error code is “2033” or quit if other errors occur. When the script was executed, the error code “2016” was returned. Error code “2016” is MQSeries error that indicates a queue permission problem when retrieving message. To correct the problem, administrator needs to reconfigure the queue permission.

MQTCL also provides a multi-level logging facility that records MQTCL messaging activities into log files. Successful activities and failure activities are logged into separate log files. Logging information such as when and what service is performed, service return status and detailed message description can be specified by 3 different logging levels.

Future Work

While message-oriented-middleware is becoming a critical piece of technologies in enterprise integration, many standard tools have been developed on this area, such as system monitoring tools to monitor MQSeries resources and transaction analysis tools to correlate distributed applications to business transactions. Some transaction tools can also provide business level information from your integrated enterprise systems to assist business decisions and support service level agreements.

One of the possible future directions for MQTCL is to build interfaces to those MQSeries standard tools. In this way, MQSeries enabled Tcl applications would be able to access and automate functionalities in those tools to generate business reports based on certain conditions.

Conclusion

With the proliferation of message-oriented-middleware technologies, it is necessary to have a Tcl extension to interface with MQSeries in order for Tcl language to play a critical role in enterprise integration. MQTCL has a rich set of features that enables Tcl applications readily to participate in sophisticated message-driven process and transactions.