

Industry-specific

QAD SOLUTIONS

Manufacturing Applications

Technical Reference Q/LinQ

Communication API Reference
Mapping API Reference
Publishing API Reference
Processing API Reference
Q/LinQ Utilities



78-0408F
MFG/PRO 9.0, eB, eB2, and eB2.1
September 2003

This document contains proprietary information that is protected by copyright. No part of this document may be reproduced, translated, or modified without the prior written consent of QAD Inc. The information contained in this document is subject to change without notice.

QAD Inc. provides this material as is and makes no warranty of any kind, expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. QAD Inc. shall not be liable for errors contained herein or for incidental or consequential damages (including lost profits) in connection with the furnishing, performance, or use of this material whether based on warranty, contract, or other legal theory.

MFG/PRO is a registered trademark of QAD Inc. QAD, QAD eQ, and the QAD logo are trademarks of QAD Inc.

Designations used by other companies to distinguish their products are often claimed as trademarks. In this document, the product names appear in initial capital or all capital letters. Contact the appropriate companies for more information regarding trademarks and registration.

Copyright © 2003 by QAD Inc.
78-0408F

QAD Inc.
6450 Via Real
Carpinteria, California 93013
Phone (805) 684-6614
Fax (805) 684-1890
<http://www.qad.com>

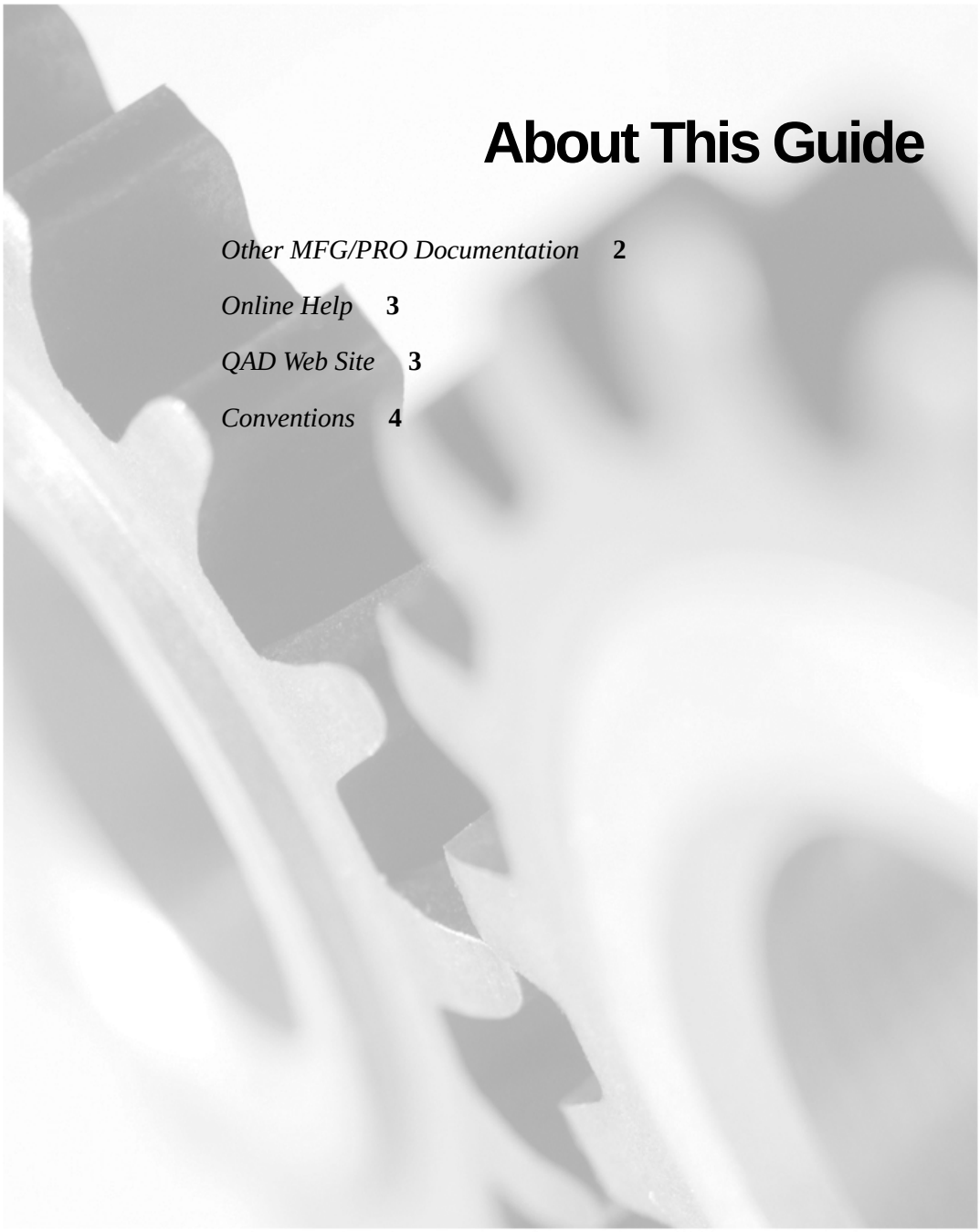
Contents

About This Guide	1
Other MFG/PRO Documentation	2
Online Help	3
QAD Web Site	3
Conventions	4
Menu References	4
Interface	5
Typographic	5
Chapter 1 Overview	7
Introduction	8
File Locations	8
Programming Conventions	9
Chapter 2 Communication API Reference	11
Overview	12
Stream API	13
Messaging API	14
MOM API	16
Stream API Controller	16
Program Decomposition: Stream API Controller Adapter	17
qqOpenSrcDest	20
qqWriteDocCtrl	24
qqWriteDocData	32

qqReadDocCtrl	33
qqReadDocData	40
qqCloseSrcDest	41
Stream API Services	42
Program Decomposition: Stream API Services, qqstrapi.p	42
qqStartSession	44
qqPutDocCtrl	46
qqPutDocData	50
qqGetDocCtrl	52
qqGetDocData	56
qqStopSession	58
Messaging API	59
Program Decomposition: Messaging API Adapter	60
Messaging API Adapter Functional Description	63
Predefined Constants and Public Static Variables	65
Return Codes	66
Using the Test Programs	67
Messaging API: Original Java Classes and Methods	69
Messaging API: Simplified Java Classes and Methods	70
C and Java Method Descriptions	72
connect	73
acceptMsg	75
receiveMsg	77
disconnect	78
isAlive	79
qqSendConnect	80
qqReceiveConnect	82
qqAckConnect	84
qqSendDocCtrl	85
qqSendDocData	86
qqReceiveDocCtrl	87
qqReceiveDocData	89
qqAckReceivedDoc	91
qqDisconnect	92
qqCheckConnect	94

CONNECTDATA and ConnectData Parameters	95
CTRLHEAD and CtrlHead Parameters	98
Chapter 3 Mapping API Reference	105
Overview	106
Direct Database Access	107
Program Decomposition: Mapping Procedure	107
Mapping Procedure Parameters	108
Mapping Procedure Code Example	111
qqMappedCode	113
qqGetRawDocCtrl	114
qqGetRawDocLine	116
qqGetRawDocElement	117
qqPutMappedDocCtrl	118
qqPutMappedDocLine	120
qqPutMappedDocElement	121
qqBuildCIMLine	122
qqLogMapException	123
Text File Dump/Load	124
Text File Formats	124
Program Decomposition: Mapping API Adapter	133
qqDumpRawDocFiles	135
qqLoadMappedDocFile	136
qqLoadMapExceptionFile	137
Chapter 4 Publishing API Reference	139
Overview	140
Program Decomposition: Publishing API Adapter	141
Triplet Format	141
XML Format	142
qqBeginInterfaceDoc	145
qqBuildDocElement	149
qqExtractRecordData	153
qqPublishInterfaceDoc	155

qqPublishXMLDoc	156
qqMappedCode	159
Chapter 5 Processing API Reference	161
Overview	162
Program Decomposition: Processing API Adapter	163
Gateway Program Parameters	164
Gateway Procedures	165
qqGetNextDocElement	166
qqGetNextDocLine	168
qqLogProcException	169
qqLogOutputMsg	170
qqMappedCode	171
Chapter 6 Q/LinQ Utilities.....	173
Check Application Registry	174
Identify Source Application	175
Map Code Values	176
Glossary.....	179
Index.....	185



About This Guide

Other MFG/PRO Documentation 2

Online Help 3

QAD Web Site 3

Conventions 4

This guide provides detailed technical information for Q/LinQ for multiple versions of MFG/PRO; see the title page for a complete list.

This information can be used by developers to build custom application program interfaces (APIs) that enable Q/LinQ to communicate with any external application, map data, and publish data to Q/LinQ from MFG/PRO applications. Developers can also build custom gateway programs to directly update the MFG/PRO database with data from external applications.

Use this book in combination with these additional resources:

- *External Interface Guide: Q/LinQ* discusses how to install and use Q/LinQ.
- *External Interface Guide: Logistics* discusses the Logistics API, which is an example of an interface built with Q/LinQ infrastructure.
- Technical details for the OAGIS BODs supported by Logistics can be found on the documentation portion of the QAD Web site, under the section on External Interfaces.

▶ See “QAD Web Site” on page 3.

Other MFG/PRO Documentation

- For software installation instructions, refer to the appropriate installation guide for your system.
- For conversion information, refer to the *Conversion Guide*.
- For instructions on navigating the Network User Interface (NetUI) and Desktop environments:
 - For 9.0, refer to *Network User Interface Guide* or *User Guide: eB Desktop*.
 - For MFG/PRO eB, see *User Guide: eB Desktop and Network User Interface*.
 - For MFG/PRO eB2 and eB2.1, see *User Guide: QAD Desktop*.
- For instructions on navigating the Windows and character environments:
 - For MFG/PRO 9.0, refer to the *User Interface Guide*.
 - For MFG/PRO eB, eB2, and eB2.1, see *User Guide Volume 1: Introduction*.

- For information on using MFG/PRO, refer to the *User Guides*. CIM Interface information changes by MFG/PRO version:
 - For MFG/PRO 9.0, see *User Guide Volume 11: Manager Functions*.
 - For later versions of MFG/PRO, see *User Guide Volume 9: Manager Functions*.
- For technical details about MFG/PRO, refer to *Entity Diagrams (File Relationships* in 9.0 and eB) and *Database Definitions*.
- For information on Q/Xtend, the QAD interoperability framework, see *Technical Reference: QXtend Inbound*.
- For information on using features that let MFG/PRO work with external applications, see the *External Interface Guides*.
- To view documents online in PDF format, see the *Documents on CD* and *Supplemental Documents on CD*.

Note MFG/PRO installation guides are not included on a CD. Printed copies are packaged with your software. Electronic copies of the latest versions are available on the QAD Web site.

Online Help

MFG/PRO has an extensive online help system. Help is available for most fields found on a screen. Procedure help is available for most programs that update the database. Most inquiries, reports, and browses do not have procedure help.

For information on using the help system for the different MFG/PRO interfaces, refer to the appropriate chapters of the books referenced in the previous section.

QAD Web Site

The QAD Web site provides a wide variety of information about the company and its products. You can access the Web site at:

<http://www.qad.com>

For MFG/PRO users with a QAD Web account, product documentation is available for viewing or downloading at:

<http://support.qad.com/documentation/>

You can register for a QAD Web account by accessing the Web site and clicking the Accounts link at the top of the screen. Your customer ID number is required. Access to certain areas is dependent on the type of agreement you have with QAD.

Most user documentation is available in two formats:

- Portable document format (PDF). PDF files can be downloaded from the QAD Web site to your computer. You can view them with the free Adobe Acrobat Reader. A link for downloading this program is also available on the QAD Web site.
- HTML. You can view user documentation through your Web browser. The documents include search tools for easily locating topics of interest.

Features also include an online solution database to help MFG/PRO users answer questions about setting up and using the product. Additionally, the QAD Web site has information about training classes and other services that can help you learn about MFG/PRO.

Conventions

Menu References

This guide applies to Q/LinQ in multiple versions of MFG/PRO starting with 9.0. A number of menus have changed between these versions. For example, the menus for Q/LinQ in 9.0 are different from those in later versions. Differences in menu numbers are noted, when necessary, using this format:

Register External Application (36.8.1.1; 36.8.1 in 9.0)

The initial menu number identifies the program in the most recent release. The second menu number applies to the release specified and any earlier releases.

Interface

MFG/PRO is available in several interfaces: Windows, character, Web browser, and an interface for object-oriented programs. To standardize presentation, the documentation uses the following conventions:

The software is available in several interfaces, depending on the release-level of MFG/PRO: Desktop or NetUI (Web browser), Windows, and character. To standardize presentation, the documentation uses the following conventions:

- MFG/PRO screen captures show the Desktop interface.
- References to keyboard commands are generic. For example, choose Go refers to:
 - The forward arrow in Desktop
 - F2 in the Windows interface
 - F1 in the character interface

In the character and Windows interfaces, the Progress status line at the bottom of a program window lists the main UI-specific keyboard commands used in that program. In Desktop, alternate commands are listed in the right-click context menu.

Typographic

This document uses the text or typographic conventions listed in the following table.

If you see:	It means:
monospaced text	A command or file name.
<i>italicized monospaced text</i>	A variable name for a value you enter as part of an operating system command; for example, <i>YourCDROMDir</i> .
indented command line	A long command that you enter as one line, although it appears in the text as two lines.
Note	Alerts the reader to exceptions or special conditions.
Important	Alerts the reader to critical information.
Warning	Used in situations where you can overwrite or corrupt data, unless you follow the instructions.

The background of the page is a grayscale photograph of several interlocking mechanical gears. The gears are of different sizes and are arranged in a way that creates a sense of depth and complexity. The lighting is soft, highlighting the metallic texture and the sharp edges of the gear teeth.

Chapter 1

Overview

This chapter provides an overview of the topics discussed in this reference and describes the conventions used in presenting information.

Introduction **8**

File Locations **8**

Programming Conventions **9**

Introduction

This book includes technical information about Q/LinQ. Using this information, developers can:

- Build custom application program interfaces (APIs) that enable Q/LinQ to communicate with any external application.
- Build custom APIs to map data.
- Publish data to Q/LinQ from MFG/PRO applications.
- Build custom gateway programs to directly update the MFG/PRO database with data from external applications.

Use the information in this book in conjunction with *External Interface Guide: Q/LinQ*, which introduces the architecture of Q/LinQ and discusses its data exchange methods. It provides details on how to set up and register programs in Q/LinQ.

The publishing API includes a method for publishing XML documents. This method can be used in conjunction with DOM Builder, a general-purpose API for developing and managing XML documents. This utility is described in detail in *Technical Reference: QXtend Inbound*. This book also contains information on setting up and using Q/LinQ to process QDocs, the XML data files generated with QXtend.

File Locations

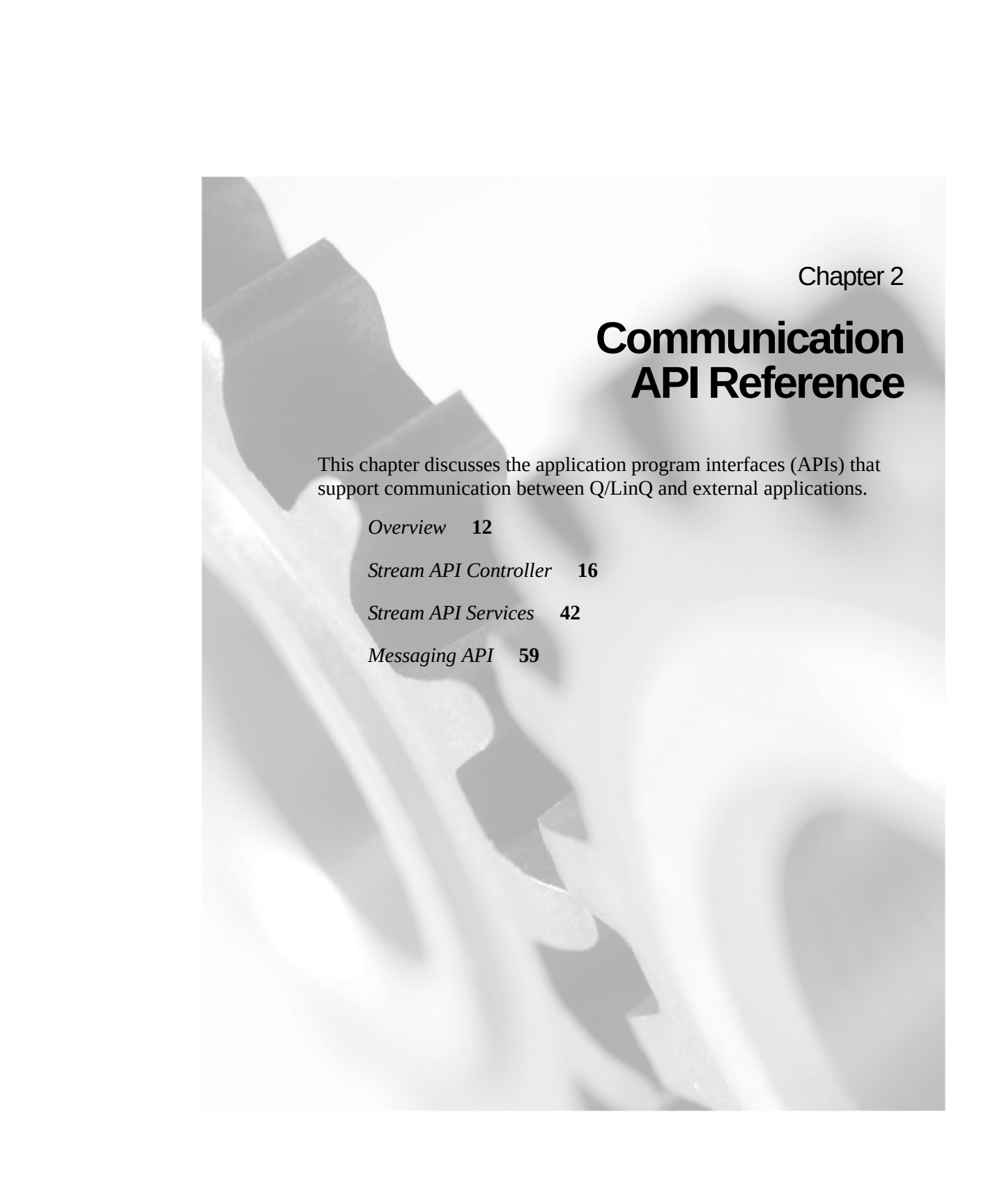
Look in *YourQlinqInstallDir*/qlinq for the following documents, library files, and source code.

Table 1.1
File Locations

Directory	Description
lib	Subdirectories specific to various operating system platforms. They contain the C static library files necessary to build adapters using the messaging API. The docs subdirectory contains javadoc HTML files describing the class interfaces making up the Java messaging API.
samples	Subdirectories with source code samples using each of the Q/LinQ APIs: publishing, mapping, messaging, stream, and processing.

Programming Conventions

- Routine, as used in this book, refers to any C function, Java method, or Progress routine.
- All required parameters in Progress, Java, and C routines must have valid data values supplied as arguments.
- Progress routines accept the undefined value (?) for parameters defined as optional. They ignore any arguments that are the undefined value.
- C functions accept an empty argument for optional parameters.
- Java methods must have valid variables for all parameters, even the optional ones.
- The C convention for function names on the UNIX operating system is lower-case letters. The Java requirement is initial capitals for words in class names (QqApi). For methods and parameters, words have initial capitals, except that the first letter of the method or parameter name is lower case (qqReadDocData). In general, the names used in this document are designed to emphasize readability. Be sure to use the proper language convention in your programs.

The background of the page is a grayscale image of several interlocking gears. The gears are of different sizes and are positioned in a way that creates a sense of depth and mechanical complexity. The lighting is soft, highlighting the teeth of the gears and creating subtle shadows.

Chapter 2

Communication API Reference

This chapter discusses the application program interfaces (APIs) that support communication between Q/LinQ and external applications.

Overview **12**

Stream API Controller **16**

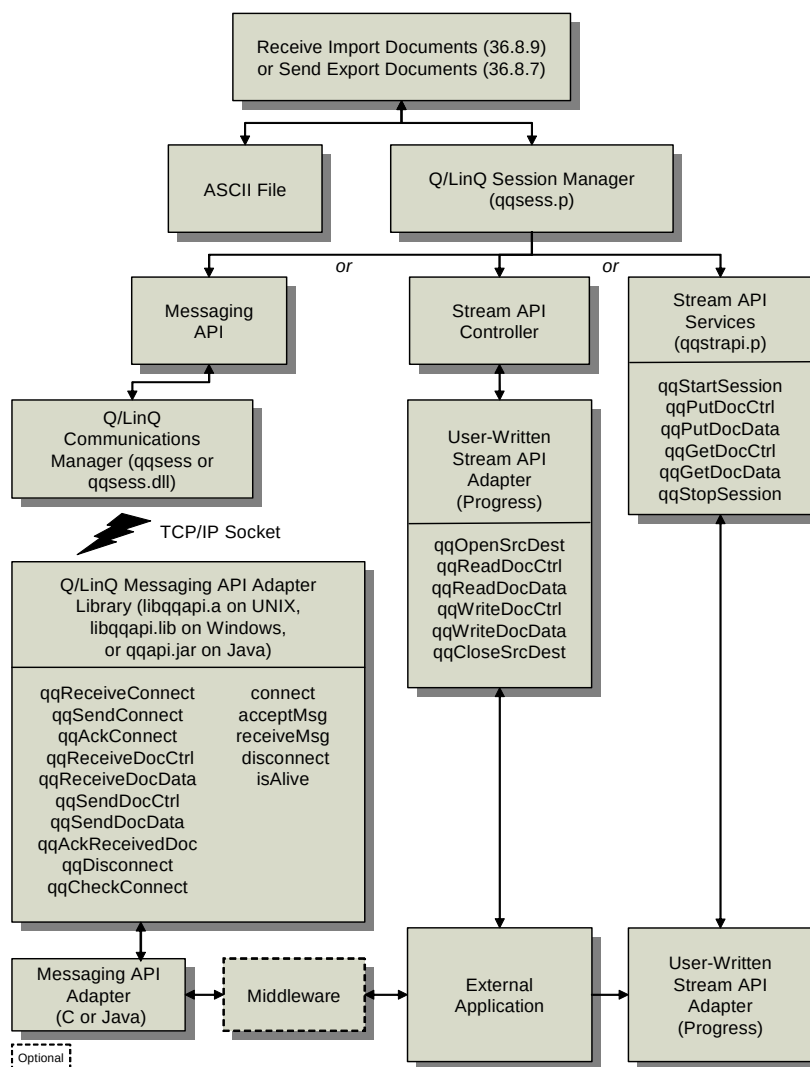
Stream API Services **42**

Messaging API **59**

Overview

The stream and messaging APIs support communication with external applications. In addition, QAD provides a messaging API adapter to the IBM message-oriented middleware (MOM) product, IBM WebSphere MQ—also known as MQSeries. Figure 2.1 shows the information flow when using the various communication APIs.

Fig. 2.1
Importing from or
Exporting to an
External
Application with a
Communication
API



Stream API

Use the stream API for interfaces that connect directly to other Progress applications or databases. Also use it for transport modes that can be handled easily by the Progress 4GL, such as writing to or reading from files that do not conform to the Q/LinQ format, or transferring information through a command-line script to an e-mail system or file transfer protocol (FTP) destination.

The stream API is particularly useful when writing to or reading from a Progress database outside MFG/PRO. For example, the MFG/PRO data synchronization functions uses the stream API to connect the Q/LinQ tables of one MFG/PRO database with the Q/LinQ tables of other MFG/PRO databases.

Note Data synchronization functions are available as a separate module in MFG/PRO 9.0, eB, and eB2. They are incorporated into Q/LinQ in MFG/PRO eB2.1.

The stream API has two sets of methods for use in different situations:

- The stream API controller includes exit points to Progress 4GL adapter programs that:
 - Connect to external applications.
 - Disconnect from external application.
 - Read import documents into Q/LinQ.
 - Write export documents out of Q/LinQ.▶ See page 17.
- The stream API services includes methods that:
 - Start a Q/LinQ interface session.
 - Stop a Q/LinQ interface session.
 - Put import documents into Q/LinQ.
 - Get export documents out of Q/LinQ.▶ See page 42.

Messaging API

Use the messaging API for direct connectivity with non-Progress applications via sockets across a TCP/IP network. The messaging API—C functions or Java methods—is most appropriate when:

- The external application is not a Progress program; for example, the IBM MQSeries message-oriented middleware.
- The programmer lacks a Progress development license or Progress 4GL skills or prefers using C or Java.

This process-to-process communications interface requires user-written import or export adapters:

- Import adapters request documents from external applications and bring them into Q/LinQ for processing.
- Export adapters receive documents from Q/LinQ and pass them to external applications.

Q/LinQ communicates with external applications using TCP sockets of type `SOCK_STREAM`. In the UNIX environment, Berkeley Software Design (BSD) sockets are used. Windows Sockets (Winsock) libraries are used in Windows environments.

Sockets support:

- Host-to-host communication
- Communication between two processes on a single host
- Communication between applications written in different programming languages

Using TCP sockets for interprocess communication shields the Q/LinQ adapters from the custom adapters; with this arrangement, they can be maintained and compiled separately.

Registering an application that uses interprocess communication with Q/LinQ requires specifying a static IP address and port number. Q/LinQ uses this information to connect to a socket and communicate through adapters with the external application. Based on how the external application is registered with Q/LinQ, Q/LinQ can actively initiate the connection or passively wait for the external application to connect. However, Q/LinQ does not start or stop external application or adapter processes such as the MOM adapter. Users must do this outside Q/LinQ.

After an export document is successfully delivered to an outside application, Q/LinQ has no control over it. Q/LinQ receives an acknowledgment that the external application received each packet of data in the document.

Note This communications acknowledgment is independent of the Q/LinQ autoacknowledgment feature. The first is a low-level call within the messaging API after transmission of each packet of data and is invisible to users. The second is sent when the entire document has been received and processed by the application, and can be seen by Q/LinQ users.

Other than the information that the autoacknowledgment returns to Q/LinQ, Q/LinQ cannot verify that an application has correctly processed a document and has no knowledge of the type of processing performed on a document.

Processing could consist of dumping the document to a text file, writing an entry in a database, or handing the document to the external application for asynchronous processing. When processing completes, the adapter can send an autoacknowledgment back via the socket to Q/LinQ, informing Q/LinQ of the processing results.

The messaging API provides a convenient, portable, and flexible means of access for single-host or local-area-network-based communications that minimizes reliance on third-party software licenses. However, it can pose security risks.

The messaging API cannot replace data communications middleware and has limitations with respect to data communications, including the following:

- No guaranteed delivery or other quality of service of options
- No access-security checking or data encryption
- No warm recovery, restart, or resend capability after network failures
- Predefined and static rather than dynamic IP addresses and ports to establish connections
- No multi-thread multiple interface connections through the same socket
- A separate dedicated port for each concurrent connection

Warning Never use the messaging API to send data outside the company firewall or across public networks without appropriate middleware, if security or reliability is a concern.

For robust data transport services in Internet, extranet, or wide-area network environments, use commercial communications packages such as MOM or e-mail systems. Build an interface between the communication package and the Q/LinQ messaging API.

MOM API

Q/LinQ provides two adapters written in Java for use with MOM:

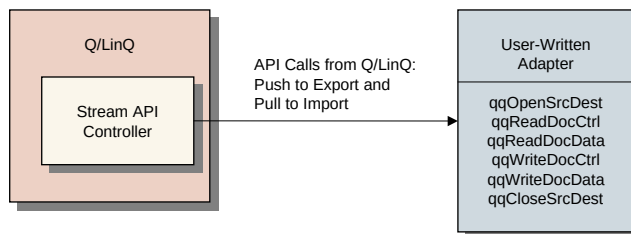
- An adapter to the MQSeries MOM, `QqMomAdapter`. It mediates communication between an MQSeries queue and a Q/LinQ session. Use MQSeries to ensure secure communications when exchanging data outside your company network.
- An adapter for use with JMS-compliant MOM products, `QqJMSAdapter`. Java Message Service (JMS) is an API from Sun Microsystems that supports messaging between computers in a network.

Tip
The JMS Adapter is not available in MFG/PRO 9.0.

Stream API Controller

The stream API controller contains a set of calls from Q/LinQ to a Progress 4GL procedure that uses the subroutines listed in Figure 2.3 and Figure 2.4. The Progress procedure is a user-written, custom adapter that implements these subroutines, providing the input and output logic that is specific to the external application. The custom adapter program must run as a persistent procedure to preserve the values of any local variables between calls.

Fig. 2.2
Stream API Controller



Use the stream API controller to initiate export or import from within Q/LinQ rather than from the external application. That is, use the stream API controller to *push* export documents out of Q/LinQ or to *pull* import documents into Q/LinQ.

To initiate document exchange from the external application, use the stream API services.

◆ See “Stream API Services” on page 42.

Program Decomposition: Stream API Controller Adapter

When importing documents, the stream API controller:

- 1 Opens a source application (`qqOpenSrcDest`).
- 2 Reads documents from it (`qqReadDocCtrl` and `qqReadDocData`).
- 3 Closes the application (`qqCloseSrcDest`).

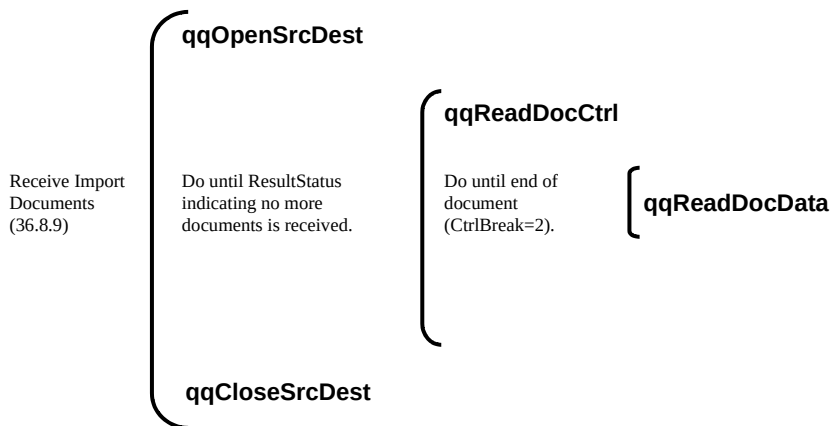


Fig. 2.3
Stream API
Controller Import
Call Logic

When exporting documents, the stream API controller:

- 1 Opens a destination application (`qqOpenSrcDest`).
- 2 Writes documents to it (`qqWriteDocCtrl` and `qqWriteDocData`).
- 3 Closes the application (`qqCloseSrcDest`).

Fig. 2.4
Stream API
Controller Export
Call Logic

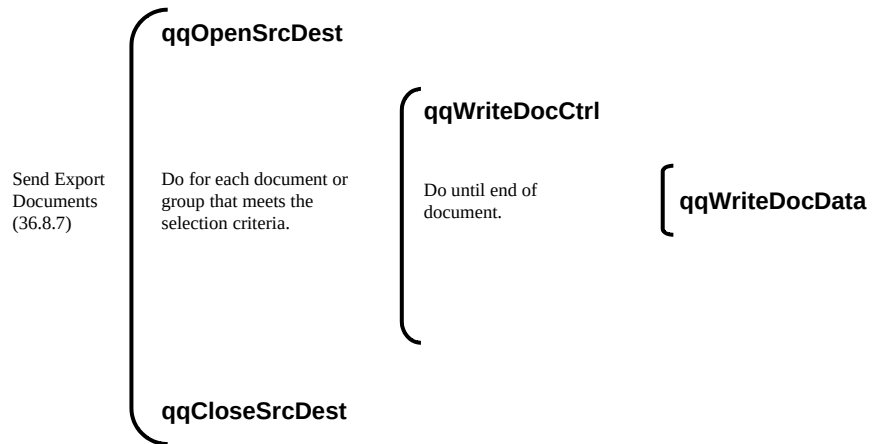
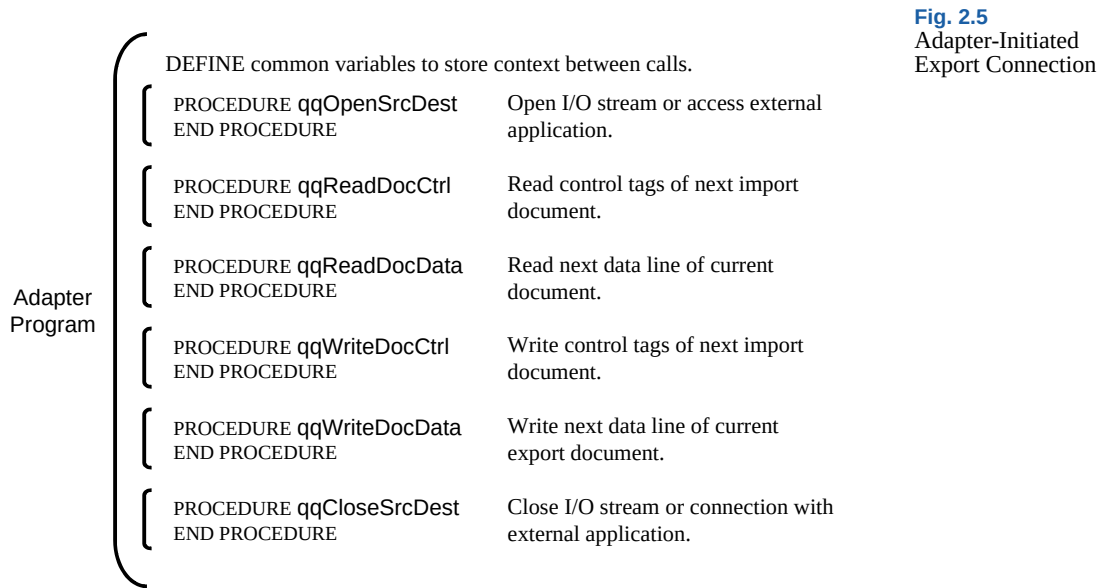


Figure 2.5 shows the general structure of a sample stream API adapter that the Q/LinQ stream API controller calls. This user-written stream API adapter fulfills:

- Q/LinQ requests to an external Progress application for import documents
- External Progress application requests to Q/LinQ for export documents

This program is not part of Q/LinQ. The information here illustrates how to use the stream API. The stream API adapter implements the internal procedures in Figure 2.3 and Figure 2.4.



qqOpenSrcDest

This API subroutine opens a connection with the external application, creates a Q/LinQ session record, and returns the Q/LinQ session number. It is called before using any of the other stream API subroutines.

A single session can either export documents from Q/LinQ or import documents into Q/LinQ, not both. Separate sessions are required for each direction of exchange.

Parameter Name	Type	Description	I/O	Req
appID	x(12)	External application to which Q/LinQ connects. The application must be registered with Q/LinQ.	I	Y
appVer	x(8)	Version of the external application referenced during interface communications. Can be used to verify version compatibility or currency before processing.	I	N
sysID	x(10)	Logical identifier for Q/LinQ used in interface communications.	I	N
sysVer	x(8)	Version of the host Q/LinQ system referenced for interface communications. Can be used to verify version compatibility or currency before processing.	I	N
accessCode	x(60)	The value—such as an authentication ID or name—required by the API to open a session with the external application. Possible uses are: <ul style="list-style-type: none"> • A non-encrypted password or security-related key. • For adapters that directly access a non-MFG/PRO database, a login ID for the database management system. • For adapters that directly access a file, the path name of the file or its directory. 	I	N
exImMode	x(1)	Indicates whether a connection is for: <ul style="list-style-type: none"> • 1. Export • 0. Import Each connection must be of either one type or the other, never both.	I	Y

qqOpenSrcDest — (Page 1 of 4)

Parameter Name	Type	Description	I/O	Req
currSessID	x(8)	Session ID of the Q/LinQ interface session that is communicating with the adapter.	I	Y
msgAckLvl	x(8)	<p>Not implemented. The Quality of Service parameter for the type of acknowledgment required for document delivery.</p> <p>Message acknowledgment level refers only to the delivery of a document. This differs from autoacknowledgment, which is based on the processing of a document by the destination application. This value overrides MOM settings by application or document.</p>	I	N
msgSync	x(8)	<p>Not implemented. Optional and relevant only to some middleware products. The Quality of Service parameter describing the level of synchronization of an exported document with respect to the destination application.</p> <p>This value is the default for interface documents when exporting with middleware. Determines the scope of a Q/LinQ send operation, which can extend to:</p> <ul style="list-style-type: none"> • Only the export of the document to the local transport system. • The delivery of the document to its destination. • The processing of the document by the receiving application. 	I	N
msgSecur	x(8)	<p>Not implemented. Relevant only to some middleware products. The Quality of Service parameter describing the level of security.</p> <p>The security level can be:</p> <ul style="list-style-type: none"> • The authentication required by the destination application to read an exported document. • The level of privacy required. • The type of encryption to be applied to the document data. 	I	N

Parameter Name	Type	Description	I/O	Req
msgPriority	x(8)	Not implemented. Priority requested for delivery of documents from the application. MOM Quality of Service parameter that overrides middleware settings by application or document.	I	N
msgRouting	x(8)	Not implemented. Network routing of a document from source to a destination (such as Queue ID). MOM Quality of Service parameter that overrides middleware settings by application or document when exporting documents using middleware. In general, it is the middleware queue, mailbox, mode, or other location to get documents from or to put them into.	I	N
msgTTL	x(8)	Not implemented. Optional. The length of time or date until which undelivered document data remains valid (such as maximum wait time or expiration date). Relevant only to some middleware products.	I	N
msgClnOrd	x(8)	Not implemented. Indicates whether the processing order of documents at the destination should match the export order of a series of documents. MOM parameter and relevant only to some middleware products.	I	N
msgPersist	x(8)	Not implemented. Relevant only to some middleware products. The Quality of Service parameter describing the time exported documents should persist inside the communications middleware after Q/LinQ sends them. The persistence time affects recoverability of the message in case of transport failure.	I	N

qqOpenSrcDest — (Page 3 of 4)

Parameter Name	Type	Description	I/O	Req
appHndl	x(16)	Handle assigned by the external application to this connection. Used for retrieving context on subsequent Q/LinQ requests.	O	N
resultStatus	integer	Indicates success or failure of the request: <ul style="list-style-type: none">• 0. Success.• <>0. Failure.	O	N

qqOpenSrcDest — (Page 4 of 4)

qqWriteDocCtrl

This API subroutine retrieves document control tags or header information for the next Q/LinQ export document marked for delivery to the external application. It is called exactly once per document before retrieving any of the document data using `qqWriteDocData`.

Parameter Name	Type	Description	I/O	Req
destAppID	x(12)	Application that Q/LinQ connects to.	I	Y
docID	integer	Identifier assigned by Q/LinQ to the document.	I	Y
docSufx	integer	Number appended by Q/LinQ to the DocID after mapping if the mapping divides the document into several documents.	I	Y
appHndl	x(16)	Handle assigned by the external application to this connection.	I	N

qqWriteDocCtrl — (Page 1 of 8)

Parameter Name	Type	Description	I/O	Req
uoWorkType	x(1)	<p>Defines how the documents in the group (see groupID) are processed in relation to each other:</p> <ul style="list-style-type: none"> Individual document (0). The group comprises one document, which is an individual unit of work. Individual documents can be processed in any order. Q/LinQ skips documents with errors. First of series (1). Marks the first document in a series. The group must be processed in the order that they were imported or exported. If one contains an error, the documents processed before it are committed. First of batch (2). Marks the first document in a batch. The group must be processed in the order that they were imported or exported. If one contains an error, none of the documents are committed. Group member (9). The document is a member of a series or batch group that was previously partially imported or exported and follows the first document. <p>Q/LinQ defines unit of work for export documents based on groupings established at the time the document was published. The external application defines it for import documents.</p>	I	Y
groupID	integer	Identifier of a set of documents that have a certain processing dependency (see uoWorkType).	I	Y
groupSeq	integer	Not implemented (0 value). Ordinal position of the document within its Q/LinQ-assigned group.	I	Y
docStd	x(8)	The published standard of the document. Required for documents that have DocStd values in Register External Application. The standard describes the structure and content of the document. Examples of standards are EDIFACT, OAGIS, and ANSI X12.	I	Y

Parameter Name	Type	Description	I/O	Req
docTyp	x(21)	<p>The type of document within the document standard after Q/LinQ maps it; for example, the 850 transaction within the ANSI X12 standard or the OAGIS BOD UPDATE_INVENTORY.</p> <p>For OAGIS document types, separate verb and noun with an underscore and capitalize it; for example, SYNC_ITEM. Required for documents that have DocType values in Register External Application.</p>	I	Y
docRev	x(8)	<p>The revision number of the document after Q/LinQ maps it. Required for documents that have DocRev values in Register External Application. In the case of an OAGIS BOD, it is the three-digit number denoting the version of the BOD.</p>	I	Y
tradPtrID	x(32)	<p>Identifies the trading partner receiving or sending the document.</p> <p>Also use this parameter to differentiate between source applications that communicate using the same middleware in MOM environments.</p> <p>For example, if MFG/PRO communicates with three external applications (A, B, and C) using a common middleware product, applications A, B, and C can be separately registered in Q/LinQ or registered together as a single middleware-based application. If registered together, distinguish A, B, and C from each other by setting their Trading Partner IDs to A, B, and C, but with the same Application ID.</p> <p>When the Q/LinQ socket-based messaging API communicates with the middleware, Q/LinQ can have a connection open to only one Application ID across any one port. The Trading Partner ID allows differentiation among applications A, B, and C; you can concurrently import documents from A, B, and C using a single port.</p>	I	N

qqWriteDocCtrl — (Page 3 of 8)

Parameter Name	Type	Description	I/O	Req
extDocKey	x(16)	A document identifier assigned by and significant to the external application; for example, an external batch, transaction, or document number. The value originates in the external application. For export documents, the MFG/PRO procedure that publishes the documents supplies the value.	I	N
mfgproSite	x(8)	Identifies the MFG/PRO site associated with the export document. Established when the document was published.	I	N
mfgproKey	x(18)	A reference that identifies the document to MFG/PRO users; for example, order-line number, item number, or customer number. This value originates in MFG/PRO. For export documents, the program that publishes the documents from MFG/PRO to Q/LinQ supplies the MFG/PRO key field value.	I	N
destProc	x(12)	Procedure to execute at the destination to process the document.	I	N
ackLvlReqd	x(1)	Indicates the type of automatic processing confirmation for Q/LinQ to receive from the external application: <ul style="list-style-type: none"> • None (0). Requests no autoacknowledgment. • Error (1). Requests an autoacknowledgment if it failed. • All (2). Requests an autoacknowledgment whether the document succeeded or failed. Autoacknowledgments are date and time stamped. Q/LinQ saves them in the document's import or export log. The OAGIS CONFIRM_BOD is a primary example of an autoacknowledgment document.	I	N

Parameter Name	Type	Description	I/O	Req
origDocId	x(8)	Identifies an original source document previously imported to Q/LinQ and passed as a reference to the sending application (for example, on an acknowledgment). This field is the Q/LinQ document ID for the referenced document.	I	N
docLng	x(2)	Language in which all the text in the document is sent, as defined by the ISO language codes.	I	N
codePg	x(15)	Identifies the code page or character set (such as iso8859-1, ibm850) used to present the document data.	I	N
dateForm	x(3)	The format of dates in the document using Progress values; for example, mdy (month, day, year), ymd, or dmy.	I	N
numForm	x(8)	Describes the format used to represent decimal numbers in the interface document, using the values supported by Progress. Possible values are: <ul style="list-style-type: none"> American. The comma separates thousands and the period is a decimal point: European. The period separates thousands and the comma is a decimal point. 	I	N
srcComp	x(10)	The name of the MFG/PRO or external application's component or module that generates the export or import documents, respectively. This is usually a major business function or module, such as AR or Receiving.	I	N
srcTask	x(10)	The task or activity in MFG/PRO or the external application that generates the export or import documents, respectively. The task is a transaction or event (for example, PostGL) that is typically a level lower than component.	I	N
srcProc	x(12)	Identifies the MFG/PRO source procedure that produced this document.	I	N

qqWriteDocCtrl — (Page 5 of 8)

Parameter Name	Type	Description	I/O	Req
srcUserId	x(8)	User ID of the person or process initiating the document. Used for audit and security access.	I	N
ctrlTot1	integer	User-defined total accumulated by Q/LinQ for multi-part documents, summed over values provided by the source application each time Q/LinQ publishes one line of the document. Can be used for such things as batch counts and reconciling with external applications.	I	N
ctrlTot2	integer	See ctrlTot1.	I	N
ctrlTot3	integer	See ctrlTot1.	I	N
msgAckLvl	x(8)	Not implemented. The Quality of Service parameter for the type of acknowledgment required for document delivery. Message acknowledgment level refers only to the delivery of a document. This differs from autoacknowledgment, which is based on the processing of a document by the destination application. This value overrides MOM settings by application or document.	I	N
msgSync	x(8)	Not implemented. Relevant only to some middleware products. The Quality of Service parameter describing the level of synchronization of an exported document with respect to the destination application. Determines the scope of a Q/LinQ send operation, which can extend to: <ul style="list-style-type: none"> • Only the export of the document to the local transport system. • The delivery of the document to its destination. • The processing of the document by the receiving application. 	I	N

Parameter Name	Type	Description	I/O	Req
msgSecur	x(8)	Not implemented. Relevant only to some middleware products. The Quality of Service parameter describing the level of security. The security level can be: <ul style="list-style-type: none"> • The authentication required by the destination application to read an exported document. • The level of privacy required. • The type of encryption to be applied to the document data. 	I	N
msgPriority	x(8)	Not implemented. Priority requested for delivery of documents from the application. MOM Quality of Service parameter that overrides middleware settings by application or document.	I	N
msgRouting	x(8)	Not implemented. Network routing of a document from source to a destination (such as Queue ID). MOM Quality of Service parameter that overrides middleware settings by application or document when exporting documents using middleware. In general, it is the middleware queue, mailbox, mode, or other location to get documents from or to put them into.	I	N
msgTTL	x(8)	Not implemented. Optional. The length of time or date until which undelivered document data remains valid (such as maximum wait time or expiration date). Relevant only to some middleware products.	I	N
msgClntOrd	x(8)	Not implemented. Indicates whether the processing order of documents at the destination should match the export order of a series of documents. MOM parameter and relevant only to some middleware products.	I	N

qqWriteDocCtrl — (Page 7 of 8)

Parameter Name	Type	Description	I/O	Req
msgPersist	x(8)	Not implemented. Relevant only to some middleware products. The Quality of Service parameter describing the time exported documents should persist inside the communications middleware after Q/LinQ sends them. The persistence time affects recoverability of the message in case of transport failure.	I	N
resultStatus	integer	Indicates success or failure of the request: <ul style="list-style-type: none"> • 0. Success. • <>0. Failure. 	O	N

qqWriteDocCtrl — (Page 8 of 8)

qqWriteDocData

This API subroutine retrieves the next line of document data for the current Q/LinQ export document marked for delivery to the external application. The `qqWriteDocCtrl` subroutine is called exactly once before calling `qqWriteDocData` repeatedly to retrieve any of the document data. If Precede Data With Tags is Yes for the export document's specification, Q/LinQ returns the document control tags before the first data lines of the document, ending the tags with a `@DATA` line.

After the last line of the data for the document has been retrieved, the next call to `qqWriteDocData` returns a control break value similar to an end-of-file value when reading text files.

Parameter Name	Type	Description	I/O	Req
<code>destAppID</code>	x(12)	Application to which a connection is requested.	I	Y
<code>docID</code>	integer	Identifier assigned by Q/LinQ to the document.	I	Y
<code>docSufx</code>	integer	Number appended by Q/LinQ to the DocID after mapping if the mapping divides the document into several documents.	I	Y
<code>appHndl</code>	x(16)	Handle assigned by the external application to this connection.	I	N
<code>ctrlBreak</code>	x(1)	<ul style="list-style-type: none"> • 2. The data represents the end of the document, that is, after the last line of data has been written. • 1. The data represents the end of the current line in the document. • 0. The data continues to the next line. 	I	Y
<code>docLength</code>	integer	Number of characters included in <code>docData</code> .	I	Y
<code>docData</code>	x(900)	A packet of data from the exported document.	I	Y
<code>resultStatus</code>	integer	Indicates success or failure of the request. <ul style="list-style-type: none"> • 0. Success. • <>0. Failure. 	O	N

qqReadDocCtrl

This API subroutine reads document control tags or header information into Q/LinQ for the next import document passed from the external application. It is called exactly once per document before putting any of the document data into Q/LinQ with `qqReadDocData`. It is followed by calls to only `qqReadDocData`, `qqReadDocCtrl` (in the case of a null document), or `qqCloseSrcDest`.

Parameter Name	Type	Description	I/O	Req
srcAppID	x(12)	External application sending the import document.	I	Y
appHndl	x(16)	Handle assigned by the external application to this connection.	I	N
resultStatus	integer	Indicates success or failure of the request. <ul style="list-style-type: none"> • 0. Success. • <>0. Failure. 	O	N

qqReadDocCtrl — (Page 1 of 7)

Parameter Name	Type	Description	I/O	Req
uowType	x(1)	<p>Defines how the documents in the group (see <code>extGroupID</code>) are processed in relation to each other:</p> <ul style="list-style-type: none"> Individual document (0). The group comprises one document, which is an individual unit of work. Individual documents can be processed in any order. Q/LinQ skips documents with errors. First of series (1). The document is the first in a series group whose members must be processed in the order that they were imported or exported. If one contains an error, the documents processed before it are committed. First of batch (2). The document is the first in a batch group whose members must be processed in the order that they were imported or exported. If one contains an error, none of the documents are committed. Group member (9). The document is a member of a series or batch group that follows the first document in the group. <p>Q/LinQ defines unit of work for export documents based on groupings established at the time the document was published. The external application defines it for import documents.</p>	O	Y
extGroupID	integer	The ID of the group containing the document in the external application. Assigned by the external application and not used by Q/LinQ.	O	Y
extGroupSeq	integer	The ordinal position within the group containing the document in the external application. It is assigned by the external application and not used by Q/LinQ.	I	N

qqReadDocCtrl — (Page 2 of 7)

Parameter Name	Type	Description	I/O	Req
docStd	x(8)	The published standard of the document. Required for documents that have DocStd values in Register External Application. The standard describes the structure and content of the document. Examples of standards are EDIFACT, OAGIS, and ANSI X12.	O	Y
docTyp	x(21)	The type of document within the document standard after Q/LinQ maps it; for example, the 850 transaction within the ANSI X12 standard or the OAGIS BOD UPDATE_INVENTORY. For OAGIS document types, separate verb and noun with an underscore and capitalize it; for example, SYNC_ITEM. Required for documents that have DocType values in Register External Application.	O	Y
docRev	x(8)	The revision number of the document after Q/LinQ maps it. Required for documents that have DocRev values in Register External Application. In the case of an OAGIS BOD, it is the three-digit number denoting the version of the BOD.	O	Y

qqReadDocCtrl — (Page 3 of 7)

Parameter Name	Type	Description	I/O	Req
tradPtrID	x(32)	<p>Identifies the trading partner receiving or sending the document.</p> <p>Also use this parameter to differentiate between source applications that communicate using the same middleware in MOM environments.</p> <p>For example, if MFG/PRO communicates with three external applications (A, B, and C) using a common middleware application, A, B, and C can be separately registered in Q/LinQ; or the three can be registered together as a single middleware-based application. If registered together, distinguish A, B, and C from each other by setting their Trading Partner IDs to A, B, and C, but with the same Application ID.</p> <p>When the Q/LinQ socket-based messaging API communicates with the middleware, Q/LinQ can have a connection open only to one Application ID across any one port. The Trading Partner ID allows differentiation among applications, and you can concurrently import documents from A, B, and C using a single port.</p>	O	N
extDockKey	x(16)	<p>A document identifier assigned by and significant to the external application; for example, an external batch, transaction, or document number. The value originates in the external application.</p> <p>For import documents, the API that publishes the documents from the external application to MFG/PRO supplies the value.</p>	O	N
mfgproSite	x(8)	Identifies the MFG/PRO site the import document is directed to.	O	N

qqReadDocCtrl — (Page 4 of 7)

Parameter Name	Type	Description	I/O	Req
mfgproKey	x(18)	A reference that identifies the document to MFG/PRO users; for example, order-line number, item number, or customer number. For import documents, the API that publishes the documents from the external application to MFG/PRO supplies the value.	O	N
destProc	x(12)	MFG/PRO procedure name to execute to process this imported document.	O	N
ackLvlReqd	x(1)	Indicates the type of automatic processing confirmation for Q/LinQ to send to or receive from the external application: <ul style="list-style-type: none"> • None (0). Sends no autoacknowledgment. • Error (1). Sends an autoacknowledgment if it failed. • All (2). Sends an autoacknowledgment whether the import or export succeeded or failed. Autoacknowledgments include a date and time stamp. Q/LinQ saves them in the document's import or export log. The OAGIS CONFIRM_BOD is a primary example of an autoacknowledgment document.	O	N
origDocId	x(8)	Identifies an original source document previously exported from Q/LinQ passed as a reference from the sending application (such as an acknowledgment). This field is the Q/LinQ document ID for the referenced document.	O	N
docLng	x(2)	Language in which all the text in the document is sent, as defined by the ISO language codes.	O	N
codePg	x(15)	Identifies the code page or character set (such as iso8859-1, ibm850) used to present the document data.	O	N

Parameter Name	Type	Description	I/O	Req
dateForm	x(3)	The format of dates in the document using Progress values; for example, mdy (month, day, year), ymd, or dmy.	I	N
numForm	x(8)	Describes the format used to represent decimal numbers in the interface document, using the values supported by Progress. Possible values are: <ul style="list-style-type: none"> American. The comma separates thousands and the period is a decimal point. European. The period separates thousands and the comma is a decimal point. 	I	N
dateCreate	nn/nn/nn	Date the document was created by the external application.	O	N
timeCreate	nn:nn:nn	Time of day the document was created by the external application.	O	N
timeZone	x(5)	The time zone of dateCreate and timeCreate , expressed as an offset from Greenwich Mean Time. For example, -0800 for Pacific Time in the US or +0100 for Berlin, Germany.	O	N
srcComp	x(10)	The name of the MFG/PRO or external application's component or module that generates the export or import documents, respectively. This is usually a major business function or module, such as AR or Receiving.	O	N
srcTask	x(10)	The name of the MFG/PRO or external application's component or module that generates the export or import documents, respectively. This is usually a major business function or module, such as AR or Receiving.	O	N
srcProc	x(12)	Identifies the external application's source procedure that produced this document.	O	N
srcUserID	x(8)	User ID of the person or process initiating the document, used for audit and security access.	O	N
ctrlTot1	integer	Not used on received documents.	O	N

qqReadDocCtrl — (Page 6 of 7)

Parameter Name	Type	Description	I/O	Req
ctrlTot2	integer	Not used on received documents.	O	N
ctrlTot3	integer	Not used on received documents.	O	N

qqReadDocCtrl — (Page 7 of 7)

qqReadDocData

This API subroutine reads the next line of document data into Q/LinQ for the current import document from the external application.

qqReadDocCtrl is called exactly once before calling this subroutine repeatedly to read the document data.

Q/LinQ attempts to read the first lines of the document as Q/LinQ document control tags until reading a @DATA line. Q/LinQ assumes that the lines are pure data if no control tag is found on the first line. Any tags provided in the data overwrite the corresponding values from the earlier qqReadDocCtrl call.

Parameter Name	Type	Description	I/O	Req
srcAppID	x(12)	External application sending the import document.	I	Y
appHndl	x(16)	Handle assigned by the external application to this connection.	I	N
resultStatus	integer	Indicates success or failure of the request. <ul style="list-style-type: none"> • 0. Success. • <>0. Failure. 	O	N
ctrlBreak	x(1)	<ul style="list-style-type: none"> • 2. The data represents the end of the document, that is, after the last line of data has been read. • 1. The data represents the end of the current line in the document. • 0. The data continues to the next line. 	O	Y
byteCnt	integer	Number of bytes in the imported data string.	O	Y
docData	x(900)	A packet of data from the imported document.	O	Y

qqCloseSrcDest

This API subroutine removes the Q/LinQ session record, then shuts down the connection with the external application. It is the only prescribed way to end a Q/LinQ-initiated stream API session.

Once this subroutine is called, qqOpenSrcDest is called again in order to import or export more documents with the stream API subroutines.

Parameter Name	Type	Description	I/O	Req
appID	x(12)	Application to disconnect from Q/LinQ.	I	Y
appHndl	x(16)	Handle assigned by the external application to this connection.	I	N
currSessID	integer	Q/LinQ identifier for the current Q/LinQ session.	I	N
resultStatus	integer	Indicates success or failure of the request. <ul style="list-style-type: none"> • 0. Success. • <>0. Failure. 	O	N

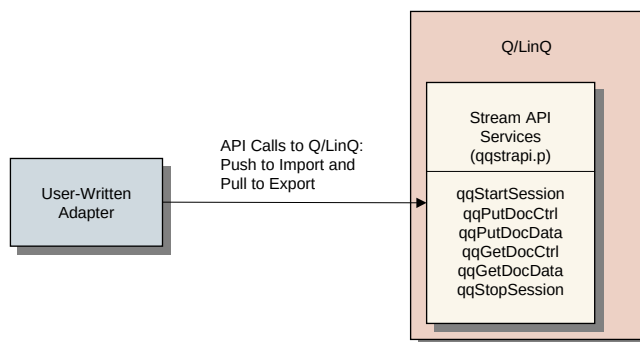
Stream API Services

▶ See Figure 2.7 and Figure 2.8.

The Q/LinQ program `qqstrapi.p` contains a set of methods—the stream API services—that can be called from a Progress 4GL procedure. The Progress procedure is a user-written, custom adapter that starts `qqstrapi.p` as a persistent procedure, then calls its methods as needed to get export documents out of Q/LinQ or put import documents into Q/LinQ. Unlike adapters used with the stream API controller, this adapter does not need to be registered in Register External Application (36.8.1.1; 36.8.1 in 9.0).

Use the stream API services to initiate export or import from the external application rather than from within Q/LinQ. That is, use the stream API services to *push* input documents into Q/LinQ or to *pull* output documents out of Q/LinQ.

Fig. 2.6
Stream API
Services



▶ See “Program Decomposition: Stream API Controller Adapter” on page 17.

To initiate document exchange from within Q/LinQ, use the stream API controller.

Program Decomposition: Stream API Services, `qqstrapi.p`

When importing documents the custom adapter program:

- 1 Starts `qqstrapi.p` as a persistent procedure.
- 2 Opens a Q/LinQ session (`qqStartSession`).
- 3 Writes documents to it (`qqPutDocCtrl` and `qqPutDocData`).
- 4 Stops the Q/LinQ session (`qqStopSession`).

5 Shuts down the persistent procedure.

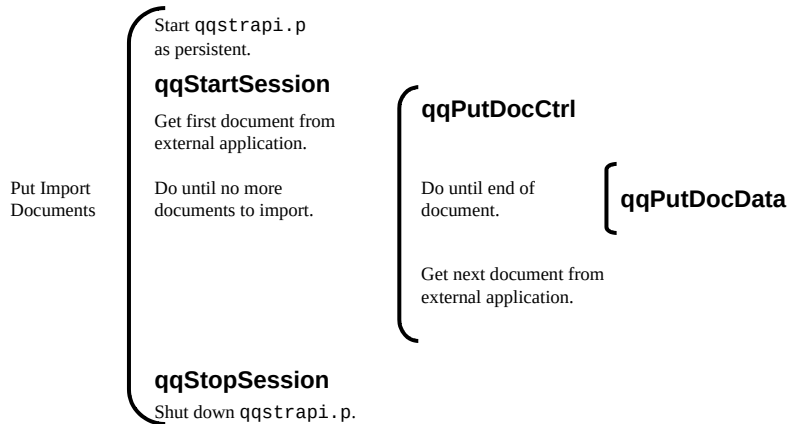


Fig. 2.7
Stream API
Services Import
Call Logic

When exporting documents, the custom adapter program:

- 1 Starts qqstrapi.p as a persistent procedure.
- 2 Opens a Q/LinQ session (qqStartSession).
- 3 Reads documents from it (qqGetDocCtrl and qqGetDocData).
- 4 Stops the Q/LinQ session (qqStopSession).
- 5 Shuts down the persistent procedure.

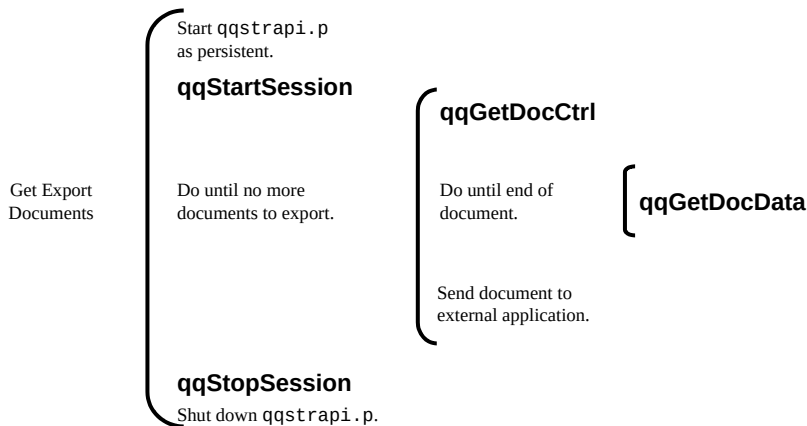


Fig. 2.8
Stream API
Services Export
Call Logic

qqStartSession

This API method creates a Q/LinQ session record and returns the Q/LinQ session number to the calling application. It must be called before using any of the other stream API methods.

A single session may either get documents out of or put documents into Q/LinQ, not both. Separate sessions are required for each direction of exchange. An adapter program can have only one open stream API session at a time.

Parameter Name	Type	Description	I/O	Req
srcDestAppID	char	External application that is the source of or destination for the documents.	I	Y
appVer	char	Version of the source or destination application, if relevant. Not presently used by Q/LinQ.	I	N
accessCode	char	A code or pathname used to obtain access to the external application. May be used in later Q/LinQ releases to authenticate the caller before starting the stream API session.	I	N
exImMode	integer	Indicates whether a connection is for: <ul style="list-style-type: none"> • 1. Export • 0. Import Each connection must be of either one type or the other, never both.	I	Y
appHndl	char	Q/LinQ-assigned session number identifying the caller's session.	O	N
resultStatus	integer	Indicates success or failure of the request. <ul style="list-style-type: none"> • 0. Success. • <>0. Failure. 	O	N

qqStartSession ResultStatus Values

Value	Description	Severity
2526	Q/LinQ control record not found.	ERROR
2476	Application # not registered.	ERROR
2754	Application # is connected for export by session #, user #.	ERROR
2756	Application # is connected for import by session#, user #.	ERROR
2464	Receiving from application # is disabled.	ERROR
2480	Sending to application # is disabled.	ERROR

qqPutDocCtrl

This API method writes document control tags or header information into Q/LinQ for the next import document passed from the calling application. It must be called exactly once per document before putting any of the document data into Q/LinQ with qqPutDocData. It is followed by calls to only qqPutDocData, qqPutDocCtrl (in the case of a null document), or qqStopSession.

Parameter Name	Type	Description	I/O	Req
srcAppID	char	Identifies the application from which the document is being passed. Not presently used.	I	N
appHndl	char	Identifies the Q/LinQ stream API session. Not presently used.	I	N
uowType	char	<p>Defines how the documents in the group are processed in relation to each other:</p> <ul style="list-style-type: none"> • Individual document (0). The group comprises one document, which is an individual unit of work. Individual documents can be processed in any order. Q/LinQ skips documents with errors. • First of series (1). Marks the first document in a series. The group must be processed in the order that they were imported or exported. If one contains an error, the documents processed before it are committed. • First of batch (2). Marks the first document in a batch. The group must be processed in the order that they were imported or exported. If one contains an error, none of the documents are committed. • Group member (9). The document is a member of a series or batch group that was previously partially imported or exported and follows the first document. <p>Q/LinQ defines unit of work for export documents based on groupings established at the time the document was published. The external application defines it for import documents.</p>	I	N

Parameter Name	Type	Description	I/O	Req
extGrpID	integer	Externally assigned identifier of the group containing the document. Not presently used.	I	N
extGrpSeq	integer	Sequence of the document within its group. Not presently used.	I	N
docStd	char	The published standard to which the document conforms.	I	N
docTyp	char	The type of document within the document standard. Normally important to describe the meaning of the document data and how to process it.	I	N
docRev	char	Revision number of the document type within its published standard.	I	N
tradPtrID	char	Identifies the trading partner sending the document.	I	N
extDocKey	char	A document identifier assigned by and significant to the external application.	I	N
mfgProSite	char	Identifies the MFG/PRO site for which the document is intended or to which it is most relevant.	I	N
mfgProKey	char	A reference identifying the document to MFG/PRO users; for example, order-line number, item number, or customer number. This value originates in MFG/PRO.	I	N
destProc	char	MFG/PRO procedure to call to process the document as an MFG/PRO transaction.	I	N
ackLvlReqd	char	Indicates the type of automatic confirmation regarding the processing of the document that the external application requests from Q/LinQ: <ul style="list-style-type: none"> • None (0). Sends or requests no autoacknowledgment. • Error (1). Sends or requests an autoacknowledgment if the document failed. • All (2). Sends or requests an autoacknowledgment whether the document succeeded or failed. 	I	N

Parameter Name	Type	Description	I/O	Req
origDocId	integer	For an autoacknowledgment, the Q/LinQ-assigned document ID for the document being acknowledged.	I	N
docLng	char	MFG/PRO-defined language code identifying the primary language in which the document data is written.	I	N
codePg	char	Identifies the code page or character set (such as iso8859-1, ibm850) used to present the document data.	I	N
dateForm	char	The format of dates in the document using Progress values; for example, mdy (month, day, year), ymd, or dmy.	I	N
numForm	char	Format of decimal numbers in the document as defined by Progress format types: <ul style="list-style-type: none"> American. The comma separates thousands and the period is a decimal point European. The period separates thousands and the comma is a decimal point 	I	N
srcComp	char	The name of the external application component or module that generated the document. This is usually a major business function or module, such as AR or Receiving.	I	N
srcTask	char	The task or activity in the external application that generates the documents. The task is a transaction or event (for example, PostGL) that is typically a level lower than component.	I	N
srcProc	char	Identifies the procedure in the external application that generated this document.	I	N
srcUserId	char	Identifies the user within the external application that caused the document to be generated.	I	N
ctrlTot1	integer	User-defined total accumulated for multi-part documents. Can be used for batch counts or hash totals. Not currently used by Q/LinQ.	I	N
ctrlTot2	dec	See ctrlTot1. Not currently in use.	I	N

qqWriteDocCtrl — (Page 3 of 4)

Parameter Name	Type	Description	I/O	Req
ctrlTot3	dec	See ctrlTot1. Not currently in use.	I	N
resultStatus	integer	Indicates success or failure of the request. <ul style="list-style-type: none"> • 0. Success. • <>0. Failure. 	O	N

qqWriteDocCtrl — (Page 4 of 4)

qqPutDocCtrl ResultStatus Values

Value	Description	Severity
2461	Interface session canceled by user #.	ERROR
2462	Interface session paused by user #.	ERROR
2527	Interface session record not found.	ERROR
4117	Session open for document export only.	ERROR
2733	Cannot continue group from previous receiving session.	ERROR
4120	No batch or series group started.	ERROR

qqPutDocData

This API method writes the next line of document data for the current import document passed to Q/LinQ from the calling application. Call `qqPutDocCtrl` exactly once before calling this method repeatedly to write the document data.

Q/LinQ attempts to read the first lines of the document as Q/LinQ document control tags until reading a `@DATA` line. Q/LinQ assumes that the lines are pure data if no control tag is found on the first line. Any tags provided in the data overwrite the corresponding values from the earlier `qqPutDocCtrl` call.

Parameter Name	Type	Description	I/O	Req
<code>srcAppID</code>	char	External application sending the import document. Not currently used.	I	N
<code>appHndl</code>	char	Identifies the Q/LinQ stream API session. Not currently used.	I	N
<code>ctrlBreak</code>	char	<ul style="list-style-type: none"> • 2. The data represents the end of the document, that is, after the last line of data has been written. • 1. The data represents the end of the current line in the document. • 0. The data continues to the next line. 	I	Y
<code>docLength</code>	integer	Number of characters included in <code>docData</code> . Not currently used.	I	Y
<code>docData</code>	char	A packet of data from the document.	I	Y
<code>resultStatus</code>	integer	Indicates success or failure of the request. <ul style="list-style-type: none"> • 0. Success. • $\neq 0$. Failure. 	O	N

qqPutDocData ResultStatus Values

Value	Description	Severity
2461	Interface session canceled by user #.	ERROR
2462	Interface session paused by user #.	ERROR
2527	Interface session record not found.	ERROR
4117	Session open for document export only.	ERROR
4121	Must import control header before data.	ERROR
2531	Unrecognized tag # in input data.	WARN
4122	Invalid control break.	WARN

qqGetDocCtrl

This API method retrieves document control tags or header information for the next Q/LinQ export document marked for delivery to the calling application. It must be called exactly once per document before retrieving any of the document data using `qqGetDocData`.

When no documents are left to retrieve, the next call to `qqGetDocCtrl` returns a result status value of 4118, signifying that no more documents are present. At this point, the session should normally be stopped using `qqStopSession`.

Parameter Name	Type	Description	I/O	Req
sestAppID	char	Identifies the application to which the document is being passed. Not currently used.	I/O	N
appHndl	char	Identifies the Q/LinQ stream API session. Not currently used.	I	N
resultStatus	integer	Indicates success or failure of the request. <ul style="list-style-type: none"> • 0. Success. • <>0. Failure. 	O	N
docID	integer	Identifier assigned by Q/LinQ to the document.	O	N
docSufx	integer	Number appended by Q/LinQ to the <code>docID</code> after mapping if the mapping divides the document into several documents.	O	N

qqGetDocCtrl — (Page 1 of 4)

Parameter Name	Type	Description	I/O	Req
uowType	char	<p>Defines how the documents in the group (see groupID) are processed in relation to each other:</p> <ul style="list-style-type: none"> • Individual document (0). The group comprises one document, which is an individual unit of work. Individual documents can be processed in any order. Q/LinQ skips documents with errors. • First of series (1). Marks the first document in a series. The group must be processed in the order that they were imported or exported. If one contains an error, the documents processed before it are committed. • First of batch (2). Marks the first document in a batch. The group must be processed in the order that they were imported or exported. If one contains an error, none of the documents are committed. • Group member (9). The document is a member of a series or batch group that was previously partially imported or exported and follows the first document. <p>Q/LinQ defines unit of work for export documents based on groupings established at the time the document was published. The external application defines it for import documents.</p>	O	N
grpID	integer	Q/LinQ-assigned identifier of the group containing the document.	O	N
grpSeq	integer	Not implemented (0 value). Ordinal position of the document within its Q/LinQ-assigned group.	O	N
docStd	char	The published standard to which the document conforms.	O	N
docTyp	char	The type of document within the document standard. Important in describing the meaning of the document data and how to process it.	O	N
docRev	char	The revision number of the document type within its published standard.	O	N

Parameter Name	Type	Description	I/O	Req
tradPtrID	char	Identifies the trading partner for which the document is intended.	O	N
extDocKey	char	A document identifier assigned by and significant to the external application.	O	N
mfgProSite	char	Identifies the MFG/PRO site from which the document originated or to which it is most relevant.	O	N
mfgProKey	char	A reference that identifies the document to MFG/PRO users; for example, order-line number, item number, or customer number.	O	N
destProc	char	Procedure name in the external application to which the document is directed.	O	N
ackLvlReqd	char	Indicates the type of automatic processing confirmation for Q/LinQ to receive from the external application: <ul style="list-style-type: none"> • None (0). Requests no autoacknowledgment. • Error (1). Requests an autoacknowledgment if the document failed. • All (2). Requests an autoacknowledgment whether the document succeeded or failed. 	O	N
origDocId	integer	For an autoacknowledgment, the Q/LinQ-assigned document ID for the document being acknowledged. For other documents, the Q/LinQ-assigned document ID.	O	N
docLng	char	MFG/PRO-defined language code identifying the primary language in which the document data is written.	O	N
codePg	char	Identifies the code page or character set (such as iso8859-1, ibm850) used to present the document data.	O	N
dateForm	char	The format of dates in the document as defined by Progress format types; for example, mdy (month, day, year), ymd, or dmy.	O	N

qqGetDocCtrl — (Page 3 of 4)

Parameter Name	Type	Description	I/O	Req
numForm	char	Describes the format used to represent decimal numbers in the interface document, using the values supported by Progress. Possible values are: <ul style="list-style-type: none"> American. The comma separates thousands and the period is a decimal point. European. The period separates thousands and the comma is a decimal point. 	O	N
srcComp	char	The name of the MFG/PRO component or module that generates the document. This is usually a major business function or module, such as AR or Receiving.	O	N
srcTask	char	The task or activity in MFG/PRO that generates the document. The task is a transaction or event (for example, PostGL) that is typically a level lower than component.	O	N
srcProc	char	Identifies the MFG/PRO source procedure that produced this document.	O	N
srcUserId	char	User ID of the person or process initiating the document. Used for audit and security access.	O	N
ctrlTot1	integer	User-defined total accumulated for multi-part documents. Can be used for such things as batch counts and hash totals.	O	N
ctrlTot2	dec	See ctrlTot1.	O	N
ctrlTot3	dec	See ctrlTot1.	O	N

qqGetDocCtrl — (Page 4 of 4)

qqGetDocCtrl ResultStatus Values

Value	Description	Severity
2461	Interface session canceled by user #.	ERROR
2462	Interface session paused by user #.	ERROR
2527	Interface session record not found.	ERROR
4116	Session open for document import only.	ERROR
4118	No more documents to retrieve.	ERROR

qqGetDocData

This API method retrieves the next line of document data for the current Q/LinQ export document marked for delivery to the calling application. The `qqGetDocCtrl` method must be called exactly once before calling `qqGetDocData` repeatedly to retrieve any of the document data. If `Precede Data With Tags` is `Yes` for the export document's specification, Q/LinQ returns the document control tags before the first data lines of the document, ending the tags with a `@DATA` line.

After the last line of the data for the document has been retrieved, the next call to `qqGetDocData` returns a control break value of 2 (end-of-document), similar to an end-of-file value when reading text files.

Parameter Name	Type	Description	I/O	Req
<code>destAppID</code>	char	Application to which a connection is requested and a documents is being passed.	I	N
<code>appHndl</code>	char	Identifies the Q/LinQ stream API session.	I	N
<code>docID</code>	integer	Identifier assigned by Q/LinQ to the document.	O	N
<code>docSufx</code>	integer	If present, the suffix in conjunction with the document ID uniquely identifies the document in Q/LinQ.	O	N
<code>ctrlBreak</code>	char	<ul style="list-style-type: none"> • 2. The data represents the end of the document, that is, after the last line of data has been written. • 1. The data represents the end of the current line in the document. • 0. The data continues to the next line. 	O	N
<code>docLength</code>	integer	Number of characters included in <code>DocData</code> .	O	N
<code>docData</code>	char	A line of data from the export document.	O	N
<code>resultStatus</code>	integer	Indicates success or failure of the request. <ul style="list-style-type: none"> • 0. Success. • $\neq 0$. Failure. 	O	N

qqGetDocData ResultStatus Values

Value	Description	Severity
2461	Interface session canceled by user #.	ERROR
2462	Interface session paused by user #.	ERROR
2527	Interface session record not found.	ERROR
4116	Session open for document import only.	ERROR
4119	Must export control header before data.	ERROR

qqStopSession

This API method removes the Q/LinQ session record, then shuts down the stream API persistent procedure. It is the only prescribed way to end a stream API session.

After calling this method, qqStartSession must be called again in order to import or export more documents with the stream API services.

Parameter Name	Type	Description	I/O	Req
appID	char	Application whose session is to stop. Not currently used.	I	N
appHndl	char	Identifies the Q/LinQ stream API session. Not currently used.	I	N
resultStatus	integer	Indicates success or failure of the request. <ul style="list-style-type: none">• 0. Success.• <>0. Failure.	O	N

Messaging API

Using a messaging API to import documents from a non-Progress external application into Q/LinQ has four steps:

- 1 Listen for connection. Q/LinQ or the external application initiates the import process by connecting to a predefined host port number and listening at the TCP socket for requests from the other party.
- 2 Make connection. The connecting party establishes communication with the remote socket, sends a message requesting a connection with the other party, and receives an acknowledgment and handle when it makes the connection. In this context, a *handle* is a temporary identifier assigned to the connection or thread by the external application.
- 3 Receive documents. The external application or MOM sends the document data in packets to Q/LinQ via the adapter that is tailored to fit the external application. The adapter calls Q/LinQ routines that pass the document attributes and data through memory buffers in the API routines.
- 4 Send acknowledgment. Q/LinQ sends a message to the external application that it successfully received or failed to receive each packet.

Note This communication acknowledgment and the one in step 4, below, has no relation to the Q/LinQ autoacknowledgment feature.

After Q/LinQ receives all document data, it stores the document in its import queue and log tables.

Exporting documents from Q/LinQ to a non-Progress external application has four steps very similar to the import steps:

- 1 Listen for connection. Q/LinQ or the external application initiates the export process.
- 2 Make connection.
- 3 Send documents.
- 4 Receive acknowledgment.

The messaging API is available to the programmer in the form of:

- Several libraries of routines:
 - ANSI C on UNIX (`libqqapi.a`)
 - ANSI C on Windows (`libqqapi.lib`)
 - Java (`qqapi.jar`)
- Include files for UNIX and Windows
- C test programs: `qqsend.c` and `qqrecv.c`
- Original Java test programs(`qqtest.jar`): `QqSend.java`, `QqRecv.java`, `wQqSend.java`, and `wQqRecv.java`
- Simplified Java test programs incorporating higher-level API calls(`qqtest.jar`): `QqSendSimple.java`, `QqRecvSimple.java`, `wQqSendSimple.java`, and `wQqRecvSimple.java`

The following four figures and the initial discussion of the messaging API structure apply to both C and Java programs. Separate sections provide details relevant to specific languages:

- Information specific to C programs begins on page 67.
- Information specific to Java programs begins on page 69.

The tables beginning on page 72 contain parameter descriptions and data types for both the C functions and data structures and the Java methods and classes.

Program Decomposition: Messaging API Adapter

Messaging adapters are not an integral part of Q/LinQ. Developers must write adapters to export documents from Q/LinQ to non-Progress applications or import documents from non-Progress applications to Q/LinQ.

Link the adapters with the QAD-provided subroutine library (`libqqapi.a`) and header file (`qqapiunix.h` for UNIX, `qqapiwin.h` for Windows, or `qqapi.jar` for Java).

The messaging adapter:

- Listens at a socket for requests for document export or import.
- Processes the requests by directly accessing the application.
- Sends replies back to Q/LinQ or the external application using the socket.

Tip

Q/LinQ does not start or stop the adapters.

The structure of a messaging adapter varies slightly based on whether Q/LinQ or the adapter initiates the connection and whether the connection is for export or import. Figure 2.9 through Figure 2.12 illustrate the logic required for each variant of the adapter program. In all four variants, the sections of the API follow the same general pattern.

The diagrams show logic applicable to the C rather than Java version of the messaging API. The Java logic is similar, but requires that API classes be instantiated prior to executing the methods. The test program code is the best guide to the details.

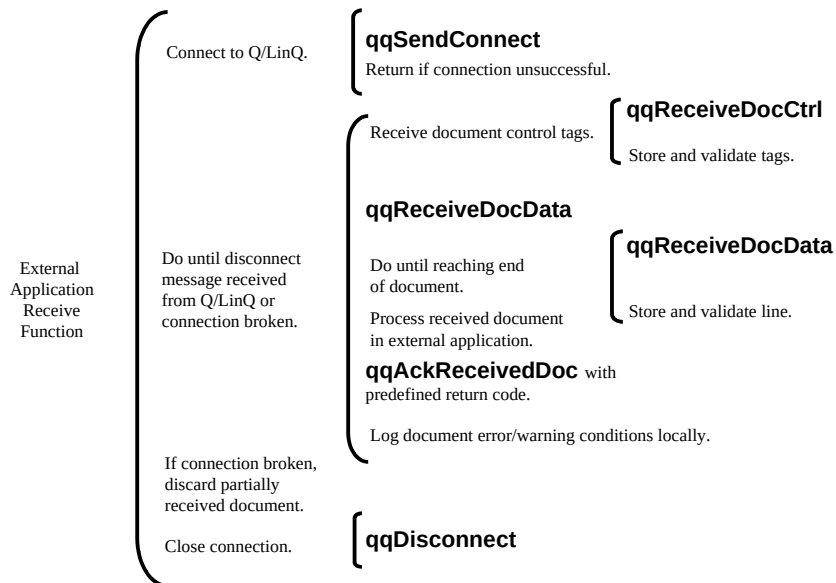


Fig. 2.9

Adapter-Initiated Export Connection

Fig. 2.10
Q/LinQ-Initiated
Export Connection

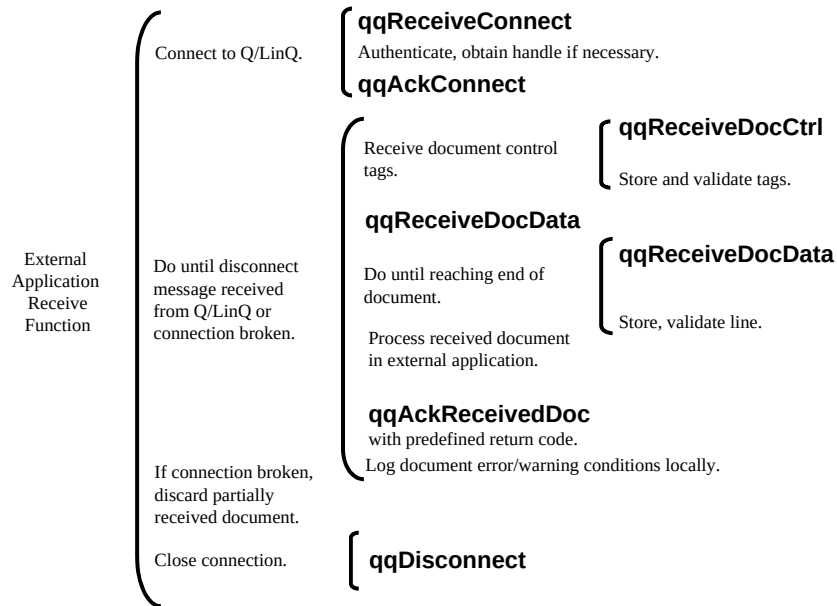
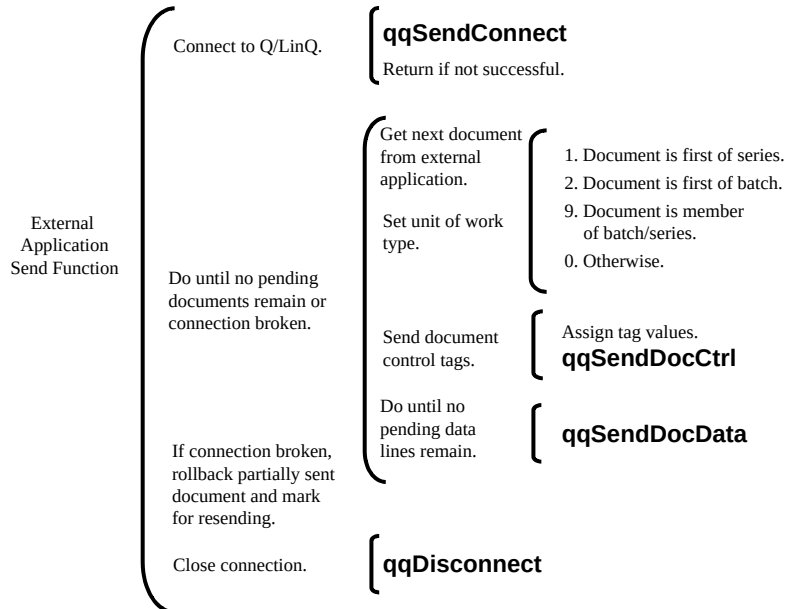


Fig. 2.11
Adapter-Initiated
Import Connection



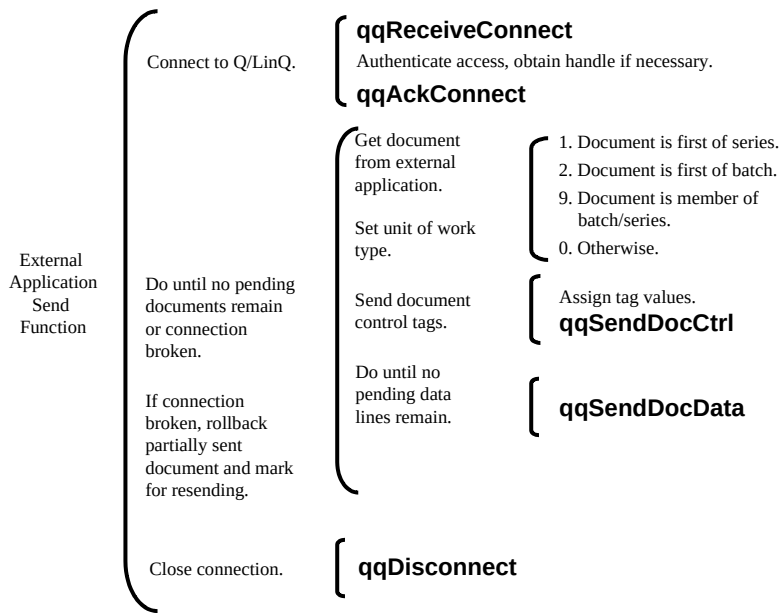


Fig. 2.12
Q/LinQ-Initiated
Import Connection

Messaging API Adapter Functional Description

Open Connection

Once started, the API either sends a connection request to Q/LinQ or waits for one from Q/LinQ. The initiating side uses `qqSendConnect`, while the receiver uses `qqReceiveConnect` and `qqAckConnect`. These routines confirm the connection and allow the external adapter to authenticate Q/LinQ and provide a handle, if one is necessary, for the session or thread.

Loop Through Documents to Send or Receive

The major loop of the adapter either sends import documents to Q/LinQ using `qqSendDocCtrl` or repeatedly looks for and receives documents exported by Q/LinQ using `qqReceiveDocCtrl`. When receiving documents from Q/LinQ, the loop looks for a `qqDisconnect` message from Q/LinQ, which terminates the loop. A control-break flag indicates that a document is the last in its group.

Note Adapters communicating with a separately running server, such as a MOM, to get documents to send to Q/LinQ should be written to poll the server at regular intervals. Using this approach, execution is not blocked indefinitely while the server has no documents to send. The adapter can also check for disconnect or reconnect requests from Q/LinQ while it is waiting for data.

Loop Through Data for Each Document Sent or Received

Within the major loop for sending or receiving documents, a minor loop sends (`qqSendDocData`) or receives (`qqReceiveDocData`) the data of each document in packets of approximately 980 characters. Documents exceeding the maximum number of characters, as well as multi-segment documents containing logical breaks, require multiple iterations through this loop. A line-continued flag in the Q/LinQ structure indicates the end of the document.

Acknowledge Each Document Received

When receiving a document, Q/LinQ requires an acknowledgment message after the final packet of data received from `qqReceiveDocData`. This acknowledgment (using `qqAckReceivedDoc`) signals Q/LinQ that the connection is still active and no data was lost. If Q/LinQ does not receive the acknowledgment, it logs the error and provides reports and notifications to assist in error recovery.

Discard or Rollback Partially Received/Sent Documents

If the connection with Q/LinQ was terminated abnormally, for example, by a system administrator interrupting the Q/LinQ session, then any document in the process of being sent or received must be rolled back to its previous state. The adapter should expect to re-receive or resend that document once communication with Q/LinQ is reestablished.

Close Connection

When the session has completed sending or receiving, socket connections must close. Regardless of who initiated the connection, the side sending documents—Q/LinQ or the adapter—initiates the disconnect. However, the receiving side can send a disconnect request to the sender with the acknowledgment message. The sending side calls `qqDisconnect`, which closes the communications link and the local socket.

The receiving side continuously checks for a `qqDisconnect` message inside its major loop and uses `qqDisconnect` to close its local socket if it detects one. After the socket disconnects, a passive adapter—one that Q/LinQ initiated communications to—should return to the open connection function and wait for another connection request from Q/LinQ or until a time-out limit is reached.

◆ See “Loop Through Documents to Send or Receive” on page 63.

Predefined Constants and Public Static Variables

Table 2.1 lists:

- For C functions, the constants defined in `qqapiunix.h` (UNIX) or `qqapiwin.h` (Windows) and used in various `qqapi` data structures and functions
- For Java methods, the public static integer variables in the `QqApi` class

Warning Do not modify any values in `qqapiunix.h` or `qqapiwin.h`. They are provided for reference only.

Name	Value	Description
IMPORT	0	Indicates the import mode of a Q/LinQ connection.
EXPORT	1	Indicates the export mode of a Q/LinQ connection.
LISTEN	0	Indicates the connection mode in which the API waits for Q/LinQ to initiate the connection.
ACTIVE	1	Indicates the connection mode in which the API initiates the connection with Q/LinQ.
QQAPI_SUCCESS	0	Return code that reports the successful execution of a Q/LinQ function.
QQAPI_FAIL	−1	Return code that reports the unsuccessful execution of a lower-level Q/LinQ function. Not used by APIs.

Table 2.1
qqapiunix and
qqapiwin
Constants and
QqApi Public
Static Integer
Variables

Return Codes

Table 2.2 lists return codes and their values in numerical order. Return codes help in interpreting and diagnosing error conditions that can occur when running adapter programs. Any error indicates data communications between O/LinQ and the adapter program either could not be established or have been broken, possibly due to network or socket problems. Users receiving these errors should contact their IS department or System Administrator to diagnose the problem. Call QAD Customer Support only after ruling out local network problems as a possible cause.

Note In Java, precede return codes with the class name since they are static attributes; for example, `QqApi.QQAPI_SUCCESS`. In C, follow the return code with an integer value.

Table 2.2
Return Codes of
qqapiunx,
qqapiwin, and
QqApi

Return Code Name	Description	Value
QQAPI_SUCCESS	Successful execution of the Q/LinQ function.	0
QQAPI_LOCAL_CONNECT_FAIL	Failed to open a local socket for Q/LinQ communications.	2471
QQAPI_INVALID_IP	Failed to establish a connection with Q/LinQ due to an invalid IP address.	2472
QQAPI_DISCONNECT_FAIL	Failed to disconnect properly from Q/LinQ.	2473
QQAPI_RECV_DOC_DATA_FAIL	Failed to receive document data from Q/LinQ.	2474
QQAPI_SEND_DOC_DATA_FAIL	Failed to send document data to Q/LinQ.	2475
QQAPI_WAIT	Failed due to a time-out while waiting for the socket to become available.	2477
QQAPI_DOC_ACK_SEND_FAIL	Failed to acknowledge that document data was received from Q/LinQ.	2624
QQAPI_DOC_ACK_RECV_FAIL	Failed to receive a valid document acknowledgment from Q/LinQ.	2625
QQAPI_LOCAL_SOCKET_FAIL	Failed to open a local socket or find the appropriate socket library functions.	2626
QQAPI_BIND_SOCKET_FAIL	Failed to bind to the local socket.	2627
QQAPI_LISTEN_FAIL	Listen mode failed.	2628

Return Code Name	Description	Value
QQAPI_CONNECT_RECV_FAIL	Failed to receive a valid connect message from Q/LinQ.	2629
QQAPI_CONNECT_SEND_FAIL	Failed to send a valid connect message to Q/LinQ.	2630
QQAPI_CONNECT_ACK_RECV_FAIL	Failed to receive a connection acknowledgment from Q/LinQ.	2631
QQAPI_CONNECT_ACK_SEND_FAIL	Failed to send a valid connection acknowledgment to Q/LinQ.	2632
QQAPI_SEND_DOC_CTRL_FAIL	Failed to send document control data to Q/LinQ.	2633
QQAPI_REMOTE_CONNECT_FAIL	Failed to establish a remote connection with Q/LinQ.	2634
QQAPI_RECV_DOC_CTRL_FAIL	Failed to receive document control data from Q/LinQ.	2635
QQAPI_SOCKET_POLL_FAIL	Failed to find a socket available for sending data.	2536

Using the Test Programs

Test communications between Q/LinQ and a custom adapter using `qqsend` and `qqrecv`, either in C or Java. By sending to `qqsend` or receiving from `qqrecv`, rather than Q/LinQ, you can determine whether communication problems originate in the adapter or the socket.

The names of the corresponding Java classes to run are `QqSend` and `QqRecv` respectively. There are also variants called `QqSendSimple` and `QqRecvSimple`, written using the higher-level simplified messaging API methods. It is recommended that `QqSendSimple` and `QqRecvSimple` be used, rather than the original versions.

The test programs can:

- Read or write data from or to text files
- Log messaging API calls
- Show erroneous return codes

While there is no need to review, compile, or understand the C source code of these programs to use them, the source code is provided as a sample to use when developing custom C or Java adapters.

C Test Programs

The C files making up the test program are as follows:

`qqsend.c` and `qqrecv.c` use the following C source modules when sending or receiving a Q/LinQ file:

- `qqgetarg.c` processes the command-line parameters in `qqsend.c` and `qqrecv.c`.
- `qqconn.c` is a generalized, high-level connect routine that calls the Q/LinQ connect routines.
- `qqprctl.c` processes Q/LinQ control-file tags and places them in the Q/LinQ control program.
- `qqvarray.c` reads the Q/LinQ file into a memory buffer for easier processing.
- `qqrdf.c` opens and reads the Q/LinQ file. Only `qqsend.c` uses it.

`qqapiunix.h` (UNIX) or `qqapiwin.h` (Windows) is the primary include file needed to write C adapters using the messaging API.

Run `qqsend` (UNIX) or `qqsend.exe` (Windows), and `qqrecv` (UNIX) or `qqrecv.exe` (Windows) from either a UNIX prompt or a Windows command prompt, as appropriate.

Java Test Programs

The Java files making up the test programs are as follows.

`QqSend.java` and `QqRecv.java` use the following Java classes when sending or receiving a Q/LinQ file:

- `wQqSend.java` contains all the important logic used by `QqSend.java`. The latter is only a command-line wrapper.
- `wQqRecv.java` contains all the important logic used by `QqRecv.java`. The latter is only a command-line wrapper.

Run `java QqSend` or `java QqRecv` from the command prompt.

The Java files making up the simplified version of the test programs are as follows:

`QqSendSimple.java` and `QqRecvSimple.java` use the following Java classes when sending or receiving a Q/LinQ file:

- `wQqSendSimple.java` contains all the important logic used by `QqSendSimple.java`. The latter is only a command-line wrapper.
- `wQqRecvSimple.java` contains all the important logic used by `QqRecvSimple.java`. The latter is only a command-line wrapper.

Run `java QqSendSimple` or `java QqRecvSimple` from the command prompt.

Messaging API: Original Java Classes and Methods

Public Class `QqApi`

This class is the original Java implementation of the messaging API for MFG/PRO, analogous to the libraries `libqqapi.a` (UNIX) and `libqqapi.lib` (Windows) that implement the C version of the messaging API functions. It implements public methods that abstract and encapsulate the Q/LinQ TCP/IP data protocol used to communicate with Q/LinQ. Adapter programs that bridge the communications from MFG/PRO to a non-Progress external application can use this API. However, it is recommended that the simplified version of the Java messaging API be used instead. The simplified classes contain higher-level methods, reducing the number of calls needed in the adapter.

Note Create one `QqApi` instance for each client-server connection with Q/LinQ, since the `QqApi` class has only a single pair of I/O streams to handle a single connection. If the same adapter program is to handle multiple connections, you must create multiple `QqApi` instances, one per connection.

`QqApi` Constructor

This constructor creates a new `QqApi` instance, which can be used to create its own TCP/IP I/O streams:

```
public QqApi(Logger log)
```

```
public QqApi(Socket socket, Logger log)
public QqApi(InputStream socketInput, OutputStream socketOutput,
Logger log)
```

Parameters of QqApi

Parameter Name	Description
socket	Local socket instance through which adapter communicates.
socketInput	InputStream instance used to send data to Q/LinQ receiver.
socketOutput	OutputStream used to receive data from Q/LinQ sender.
log	Logger instance for log file or display messages.

Public Class Logger

This is a generic log file utility class. It creates and writes to simple output (log) files.

Although this class supports multiple Logger instances pointing to the same log file, you run the risk of interference if you have multiple instances write to the same log file. Besides being unnecessary, this practice can result in confusing code.

Messaging API: Simplified Java Classes and Methods

Public Class QqApi, QqConnectionActive, QqConnectionPassive

These classes are simplified implementations of the Java messaging API classes, replacing the `QqApi` class in the original Java implementation. Like the originals, they implement public methods that abstract and encapsulate the Q/LinQ TCP/IP data protocol used to communicate with Q/LinQ. However, they are constructed as higher-level classes using the methods of the `QqApi` class, and allow adapters to be written using fewer calls and simpler logic than the originals.

- `QqConnectionActive` supports connections in which the adapter program initiates the connection with Q/LinQ.
- `QqConnectionPassive` supports connections in which the adapter program runs as a passive listener.

Most Java adapter programs that bridge the communications from MFG/PRO to a non-Progress external application should use this API.

Note Create either a `QqConnectionActive` or `QqConnectionPassive` instance for each client-server connection with Q/LinQ. If the same adapter program is to handle multiple connections, you must create multiple `QqConnectionActive` and/or `QqConnectionPassive` instances, one per connection.

QqConnectionActive and QqConnectionPassive Constructors

These constructors create a new `QqApi` instance for either an active or passive connection, which can be used to create its own TCP/IP I/O streams:

```
public QqConnectionActive(String appId, Logger log, String ipAddr,
int portNum, int pollWait)

public QqConnectionActive(ConnectData connectData, Logger log,
String ipAddr, int portNum, int pollWait)

public QqConnectionPassive(Logger log, int portNum, int pollWait)

public QqConnectionActive(Logger log, Socket socket)
```

Parameters of QqConnectionActive, QqConnectionPassive

Parameter Name	Description
appId	Identifies the connection through which Q/LinQ communicates with the adapter.
connectData	Q/LinQ connection parameters (see “CONNECTDATA and ConnectData Parameters” on page 95).
socket	Local socket instance through which the adapter communicates.
ipAddr	Pointer to IP address on the host where Q/LinQ resides.
portNum	Port number on the Q/LinQ host for the adapter on the local host to use for communication with Q/LinQ.
pollWait	Maximum number of seconds to wait for an acknowledgment message from Q/LinQ after sending each packet of document data.
log	Logger instance for log file or display messages.

Public Interface iQqSender

This interface should be implemented by all adapters responsible for receiving documents from Q/LinQ in the form of messages and sending them to other destinations. It consists of a single method `forwardMsg()`.

forwardMsg

This method is responsible for sending a Q/LinQ document to outside destinations as a message:

```
public boolean forwardMsg(byte[] msgData, CtrlHead ctrlHead)
```

Parameters of forwardMsg

Parameter Name	Description
msgData	The content of the message in ASCII format.
ctrlHead	The Q/LinQ document control tags associated with the message.

C and Java Method Descriptions

The following pages provide routine descriptions, language syntax, parameter listings, and return codes for both C functions and data structures and the Java methods and classes:

- connect 73
- acceptMsg 75
- receiveMsg 77
- disconnect 78
- isAlive 79
- qqSendConnect 80
- qqReceiveConnect 82
- qqAckConnect 84
- qqSendDocCtrl 85
- qqSendDocData 86
- qqReceiveDocCtrl 87
- qqReceiveDocData 89

- The CONNECTDATA data structure (C) and ConnectData class (Java) contain all the information necessary to establish a connection between an external application and Q/LinQ. Both `qqSendConnect` and `qqReceiveConnect` use it. 95
- `qqDisconnect` 92
- `qqCheckConnect` 94
- CONNECTDATA and ConnectData Parameters 95
- The CTRLHEAD data structure (C) and CtrlHead class (Java) contain all the control information about an interface document being sent to or received from Q/LinQ. It is used by `qqSendDocCtrl` and `qqReceiveDocCtrl`. A CTRLHEAD structure or CtrlHead object is required for every Q/linQ document. 98

connect

This method is contained in the simplified Java messaging API classes. It creates the connection with Q/LinQ: opening local and remote sockets, contacting and acknowledging Q/LinQ taking either the caller (active) or listener (passive) role. It takes the place of the `qqSendConnect` (page 80), `qqReceiveConnect` (page 82), and `qqAckConnect` (page 84) methods in the original messaging API.

Java Syntax

```
public synchronized String connect(int msgDirection,
                                   int connectSleep)
    throws QqException

public synchronized String connect(int msgDirection)
    throws QqException
```

Parameters

Parameter Name	Description	I/O	Req	C Type	Java Class
msgDirection	The direction messages will be passed, from the adapter's perspective. Values are: <ul style="list-style-type: none"> <code>QqConnection.INBOUND</code> <code>QqConnection.OUTBOUND</code> 	I	Y		int
connectSleep	Time in seconds to wait between each of three connection attempts.	I	N		int

Return Codes

Type	Description
String	Application ID of connection.

Logged Exception Codes

Exception Code Name	Description	Value
QQAPI_SUCCESS	Successful connection.	0
QQAPI_LOCAL_CONNECT_FAIL	Failed to open a local socket for Q/LinQ communications.	2471
QQAPI_INVALID_IP	Failed to establish a connection with Q/LinQ due to an invalid IP address.	2472
QQAPI_WAIT	Time-out waiting for the socket to become available.	2477
QQAPI_LOCAL_SOCKET_FAIL	Failed to open a local socket or find the appropriate socket library functions.	2626
QQAPI_BIND_SOCKET_FAIL	Failed to bind to the local socket. Often indicates that the socket was closed improperly in a previous session, preventing the same port from being used while resources are being restored to the system. This error may disappear after a few minutes have elapsed.	2627

Exception Code Name	Description	Value
QQAPI_LISTEN_FAIL	Listen mode failed.	2628
QQAPI_CONNECT_SEND_FAIL	Failed to send a valid connect message to Q/LinQ.	2630
QQAPI_REMOTE_CONNECT_FAIL	Failed to establish a connection with Q/LinQ.	2634
QQAPI_CONNECT_ACK_RECV_FAIL	Failed to receive a connection acknowledgment from Q/LinQ.	2631

acceptMsg

This method is contained in the simplified Java messaging API classes. It sends a message to Q/LinQ, taking the place of the `qqSendDocCtrl` and `qqSendDocData` methods in the original messaging API.

Java Syntax

```
public synchronized void acceptMsg(byte[] data)
    throws QqDisconnectedException, QqException
public synchronized void acceptMsg(byte[] data,
    String tradPtrId)
    throws QqDisconnectedException, QqException
public synchronized void acceptMsg(byte[] data,
    CtrlHead ctrlHead)
    throws QqDisconnectedException, QqException
```

Parameters

Parameter Name	Description	I/O	Req	C Type	Java Class
data	The data to send to Q/LinQ.	I	Y		byte[]
tradPtrId	The trading partner ID identifying the sender of the document.	I	N		String
ctrlHead	An object containing the document control tags or header that Q/LinQ will use.	I	N		CtrlHead

Return Codes

Type	Description
	None

Logged Exception Codes

Exception Code Name	Description	Value
QQAPI_SUCCESS	Successful connection.	0
QQAPI_SEND_DOC_DATA_FAIL	Failed to send document data to Q/LinQ.	2475
QQAPI_DOC_ACK_RECV_FAIL	Failed to receive a valid document acknowledgment from Q/LinQ.	2625
QQAPI_SEND_DOC_CTRL_FAIL	Failed to send document control data to Q/LinQ.	2633

receiveMsg

This method is contained in the simplified Java messaging API classes. It receives a message from Q/LinQ, taking the place of the `qqReceiveDocCtrl` and `qqReceiveDocData` methods in the original messaging API.

Java Syntax

```
public synchronized void receiveMsg(iQqSender msgSender)
    throws QNoDocumentException, QqDisconnectedException,
    QqException
```

Parameters

Parameter Name	Description	I/O	Req	C Type	Java Class
msgSender	The object that will handle the message received from Q/LinQ. The <code>iQqSender.forwardMsg()</code> method of the object is called for each message received. Typically, the adapter itself would be the <code>msgSender</code> object.	I	Y		iQqSender

Return Codes

Type	Description
	None

Logged Exception Codes

Exception Code Name	Description	Value
QQAPI_SUCCESS	Successful connection.	0
QQAPI_RECV_DOC_DATA_FAIL	Failed to receive document data from Q/LinQ.	2474
QQAPI_RECV_DOC_CTRL_FAIL	Failed to receive data from Q/LinQ.	2635
QQAPI_DISCONNECT	Q/LinQ has disconnected.	-9999

disconnect

This method is contained in the simplified Java messaging API classes. It terminates the connection with Q/LinQ, closing local and remote sockets gracefully. It takes the place of the `qqDisconnect` method in the original messaging API.

Java Syntax

```
public synchronized void disconnect()  
    throws QqException
```

Parameters

Parameter Name	Description	I/O	Req	C Type	Java Class
	None				

Return Codes

Type	Description
	None

Logged Exception Codes

Exception Code Name	Description	Value
QQAPI_SUCCESS	Successful connection.	0
QQAPI_DISCONNECT_FAIL	Failed to disconnect properly from Q/linQ.	2473

isAlive

This method is contained in the simplified Java messaging API classes. It checks the status of the connection with Q/LinQ, taking the place of the `qqCheckConnect` method in the original messaging API.

Java Syntax

```
public synchronized boolean isAlive()
```

Parameters

Parameter Name	Description	I/O	Req	C Type	Java Class
	None				

Return Codes

Type	Description
boolean	Returns true if connection to Q/LinQ is still present; otherwise, false.

Logged Exception Codes

Exception Code Name	Description	Value
	None	0

▶ See “connect” on page 73 for the simplified Java class.

qqSendConnect

This routine opens a socket on the local machine and requests a connection with Q/LinQ when the external application is registered to initiate communications. When connected, Q/LinQ responds with an acknowledgment, and this routine opens the session for further communications.

C Syntax

```
int qqSendConnect(char *ipaddr, QQINT PortNo, int PollWait, connectdata
*cdata);
```

Java Syntax

```
public int qqSendConnect(String ipAddr,
                        int portNo,
                        int pollWait,
                        ConnectData cdata)
```

Parameters

Parameter Name	Description	I/O	Req	C Type	Java Class
ipAddr	Pointer to IP address on the host where Q/LinQ resides.	I	Y	char *	String
portNo	Port number on the Q/LinQ host for the adapter on the local host to use for communication with Q/LinQ.	I	Y	QQINT	int
pollwait	Maximum number of seconds to wait for an acknowledgment message from Q/LinQ after sending each packet of document data.	I	Y	int	int
cdata	Q/LinQ connection parameters (see “CONNECTDATA and ConnectData Parameters” on page 95).	I	Y	connectdata *	ConnectData

Return Codes

Return Code Name	Description	Value
QQAPI_SUCCESS	Successful connection.	0
QQAPI_LOCAL_CONNECT_FAIL	Failed to open a local socket for Q/LinQ communications.	2471
QQAPI_INVALID_IP	Failed to establish a connection with Q/LinQ due to an invalid IP address.	2472
QQAPI_WAIT	Time-out waiting for socket to become available	2477
QQAPI_CONNECT_SEND_FAIL	Failed to send a valid connect message to Q/LinQ.	2630
QQAPI_REMOTE_CONNECT_FAIL	Failed to establish a connection with Q/LinQ.	2634

qqReceiveConnect

This routine opens a socket on the local machine and waits for a connection request from Q/LinQ when Q/LinQ initiates communications.

See page 84.

When connected, the adapter program should verify the access code provided by Q/LinQ and use qqAckConnect to notify Q/LinQ of the connection.

C Syntax

```
SCOPE int qqReceiveConnect(QQINT PortNo,int PollWait,connectdata
*cdata);
```

Java Syntax

```
public int qqReceiveConnect(int portNo,
                           int pollwait,
                           ConnectData cdata)
```

Parameters

Parameter Name	Description	I/O	Req	C Type	Java Class
portNo	Port number on the local host to be used by the application for communication with Q/LinQ.	I	N	QQINT	int
pollwait	After Q/LinQ connects with the external application, the maximum number of seconds to wait for an acknowledgment message from Q/LinQ after sending or receiving a packet of document data. When the external application initiates the connection, this timeout limit is not used.	I	N	int	int
cData	Q/LinQ connection parameters (see “CONNECTDATA and ConnectData Parameters” on page 95.		N	connectdata *	ConnectData

Return Codes

Return Code Name	Description	Value
QQAPI_SUCCESS	Connection to remote client successful.	0
QQAPI_LOCAL_CONNECT_FAIL	Failed to open a local socket for Q/LinQ communications.	2471
QQAPI_WAIT	Time-out waiting for the socket to become available.	2477
QQAPI_LOCAL_SOCKET_FAIL	Failed to open a local socket or find the appropriate socket library functions.	2626
QQAPI_BIND_SOCKET_FAIL	Failed to bind to the local socket. Often indicates that the socket was closed improperly in a previous session, preventing the same port from being used while resources are being restored to the system. This error may disappear after a few minutes have elapsed.	2627
QQAPI_LISTEN_FAIL	Listen mode failed.	2628
QQAPI_CONNECT_ACK_RECV_FAIL	Failed to receive a connection acknowledgment from Q/LinQ.	2631
QQAPI_REMOTE_CONNECT_FAIL	Failed to establish a remote connection with Q/LinQ.	2634

qqAckConnect

▶ See “connect” on page 73 for details on the simplified Java class.

This routine answers a connection request from Q/LinQ when Q/LinQ initiates communications. It can also authenticate Q/LinQ and pass it a handle for addressing the external application in subsequent calls.

C Syntax

```
SCOPE int qqAckConnect(int ConnStat,char *AppHndl);
```

Java Syntax

```
public int qqAckConnect(int connStat,  
                        byte appHndl[])
```

Parameters

Parameter Name	Description	I/O	Req	C Type	Java Class
connStat	Indicates whether or not a valid connection has been made: <ul style="list-style-type: none"> • 0, if connected. • Otherwise, the return code of an adapter-defined exception condition. 	I	Y	int	int
appHndl	Handle assigned by the external application to this connection or thread and used for subsequent calls.	I	N	char *	byte

Return Codes

Return Code Name	Description	Value
QQAPI_SUCCESS	Connection accepted and valid access code received.	0
QQAPI_WAIT	Time-out waiting for the socket to become available.	2477
QQAPI_CONNECT_ACK_SEND_FAIL	Failed to send a valid connection acknowledgment to Q/LinQ.	2634

qqSendDocCtrl

This routine sends control information about a document being sent to Q/LinQ. It must follow the establishment of an export connection with Q/LinQ using qqAckConnect (see page 98) and precede the sending of document data to Q/LinQ using qqSendDocData (see page 86).

▶ See “acceptMsg” on page 75 for details on the simplified Java class.

Note When resending documents to Q/LinQ after an abnormal loss of communication, the adapter should first resend the document that was being sent when communication was lost, then continue with any subsequent documents in the same group.

C Syntax

```
SCOPE int qqSendDocCtrl(ctrlhead *CtrlSend);
```

Java Syntax

```
public int qqSendDocCtrl(CtrlHead ctrlSend)
```

Parameters

Parameter Name	Description	I/O	Req	C Type	Java Class
ctrlSend	Pointer to the control data regarding the interface document being sent to Q/LinQ. See “CTRLHEAD or CtrlHead Parameters” on page 87).	I	N	ctrlhead *	CtrlHead

Return Codes

Return Code Name	Description	Value
QQAPI_SUCCESS	Control data sent to Q/LinQ successfully.	0
QQAPI_DOC_ACK_RECV_FAIL	Failed to receive acknowledgment from receiver that document information was received.	2625
QQAPI_SEND_DOC_CTRL_FAIL	Failed to send document control data to Q/LinQ.	2633

qqSendDocData

See “acceptMsg” on page 75 for details on the simplified Java class.

This routine sends to Q/LinQ the ASCII or extended ASCII data making up the interface document. It must follow the sending of document control information to Q/LinQ using qqSendDocCtrl (see page 85).

C Syntax

```
SCOPE int qqSendDocData(char *CtrlBreak,int ByteCnt,char *DocData);
```

Java Syntax

```
public int qqSendDocData(String ctrlBreak,
                        int byteCnt,
                        byte docData
```

Parameters

Parameter Name	Description	I/O	Req	C Type	Java Class
ctrlBreak	<ul style="list-style-type: none"> 2. The data represents the end of the document, after having sent the last line of data. 1. The data represents the end of the current line in the document. 0. The data continues to the next line. 	I	Y	char *	String
byteCnt	Number of bytes in the data string to be sent to Q/LinQ.	I	Y	int	int
docData	A pointer to a stream of data of ByteCnt length from the requested document.	I	Y	char *	byte

Return Codes

Return Code Name	Description	Value
QQAPI_SUCCESS	Data sent to Q/LinQ successfully.	0
QQAPI_SEND_DOC_DATA_FAIL	Failed to send document data to Q/LinQ.	2475
QQAPI_DOC_ACK_RECV_FAIL	Failed to receive a valid document acknowledgment from Q/LinQ.	2625

qqReceiveDocCtrl

This routine receives control information about a document being sent from Q/LinQ. It must follow the establishment of an import connection with Q/LinQ using qqSendDocData (see page 86), and precede the receiving of document data from Q/LinQ using qqReceiveDocData (see page 89).

See “receiveMsg” on page 77 for details on the simplified Java class.

C Syntax

```
SCOPE int qqReceiveDocCtrl(ctrlhead *CtrlRecv);
```

Java Syntax

```
public int qqReceiveDocCtrl(CtrlHead ctrlRecv)
```

Parameters

Parameter Name	Description	I/O	Req	C Type	Java Class
ctrlRecv	Control data regarding the interface document being received from Q/LinQ. See “The CTRLHEAD data structure (C) and CtrlHead class (Java) contain all the control information about an interface document being sent to or received from Q/LinQ. It is used by qqSendDocCtrl and qqReceiveDocCtrl. A CTRLHEAD structure or CtrlHead object is required for every Q/LinQ document.” on page 98.	O	N	ctrlhead *	CtrlHead

Return Codes

Return Code Name	Description	Value
QQAPI_SUCCESS	Data received successfully.	0

Return Code Name	Description	Value
QQAPI_RECV_DOC_CTRL_FAIL	Failed to receive data from Q/LinQ.	2635
QQAPI_DISCONNECT	Q/LinQ has disconnected.	-9999

qqReceiveDocData

This routine receives from Q/LinQ the ASCII or extended ASCII data making up the interface document. It must follow the receiving of document control information from Q/LinQ using qqReceiveDocCtrl (see page 87).

See “receiveMsg” on page 77 for details on the simplified Java class.

C Syntax

```
SCOPE char *qqReceiveDocData(char *Method, char *CtrlBreak, int
*ByteCnt, int *ResultStatus)
```

Java Syntax

```
public byte[] qqReceiveDocData(StringBuffer method,
                               StringBuffer ctrlBreak,
                               int byteCnt,
                               int resultStatus)
```

Parameters

Parameter Name	Description	I/O	Req	C Type	Java Class
method	The type of message sent by Q/LinQ that has been received by the adapter program.	O	N	char *	StringBuffer
ctrlBreak	<ul style="list-style-type: none"> 2. The data represents the end of the document, after having received the last line of data. 1. The data represents the end of the current line in the document. 0. The data continues to the next line. 	O	Y	char *	StringBuffer
byteCnt	Number of bytes in the data string received.	O	Y	int *	int
resultStatus	Return code describing the results of the method call.	O	N	int *	int

Return Codes

The `ResultStatus` parameter returns a character pointer of `*ByteCnt` length. It can have the following values after the routine executes.

Return Code Name	Description	Value
QQAPI_SUCCESS	Success.	0
QQAPI_RECV_DOC_DATA_FAIL	Failed to receive document data from Q/LinQ.	2474
QQAPI_DISCONNECT	Q/LinQ has disconnected.	-9999

qqAckReceivedDoc

This routine sends an acknowledgment to Q/LinQ after the external application receives the final packet of data for a document using qqReceiveDocData (see page 89). This signals Q/LinQ that the external application properly received the entire document and that Q/LinQ can now send more.

C Syntax

```
SCOPE int qqAckReceivedDoc(int AckValue);
```

Java Syntax

```
public int qqAckReceivedDoc(int ackValue)
```

Parameters

Parameter Name	Description	I/O	Req	C Type	Java Class
ackValue	A QQAPI_SUCCESS indicates the external application successfully received the document data; a QQAPI_DOC_ACK_SEND_FAIL, that it failed to receive the data.	I	N	int	int

Return Codes

Return Code Name	Description	Value
QQAPI_SUCCESS	Success.	0
QQAPI_WAIT	Failed due to a time-out while waiting for the socket to become available.	2477
QQAPI_DOC_ACK_SEND_FAIL	Failed to acknowledge that document data was received from Q/LinQ.	2624

qqDisconnect

▶ See “disconnect” on page 78 for details on the simplified Java class.

This routine closes the connection with Q/LinQ and the communications socket on the local machine when the external application sends interface documents to Q/LinQ. Use it to:

- End a Q/LinQ-initiated session.
- Terminate connections after a fatal error, such as the loss of data communication between the adapter and external application or Q/LinQ. When data communication is lost, the routine attempts to signal Q/LinQ that previously sent or received documents must be resent or re-received, then closes the socket so that a new connection can be established.

C Syntax

```
SCOPE int qqDisconnect(int ExImMode, char *AppID, char *AppHndl);
```

Java Syntax

```
public int qqDisconnect(int exImMode,  
                        String appId,  
                        String appHndl)
```

Parameters

Parameter Name	Description	I/O	Req	C Type	Java Class
exImMode	<ul style="list-style-type: none"> • Use <code>qqapiunix.h</code> (UNIX) or <code>qqapiwin.h</code> (Windows) parameter import when disconnecting after importing documents to Q/LinQ. • Use <code>qqapiunix.h</code> or <code>qqapiwin.h</code> parameter export after finishing an export from Q/LinQ. 	I	Y	int	int
appId	Identifies the Q/LinQ system being disconnected. Currently not used.	I	Y	char *	String
appHndl	Handle assigned by Q/LinQ to this connection. Currently not used.	I	Y	char *	String

Return Codes

Return Code Name	Description	Value
QQAPI_SUCCESS	Successfully disconnected from Q/LinQ.	0
QQAPI_DISCONNECT_FAIL	Failed to disconnect properly from Q/LinQ.	2473

▶ See “isAlive” on page 79 for details on the simplified Java class.

qqCheckConnect

This routine checks whether Q/LinQ is still connected to the adapter. It can be used to verify that Q/LinQ is still listening before sending another document from the external application to Q/LinQ.

C Syntax

Not applicable; Java only.

Java Syntax

```
public int qqCheckConnect();
```

Parameters

None

Return Codes

Return Code Name	Description	Value
QQAPI_SUCCESS	Q/LinQ is connected.	0
QQAPI_CONNECT_RECV_FAIL	Error occurred during the check.	2629
QQAPI_DISCONNECT	Q/LinQ is not connected.	-9999

CONNECTDATA and ConnectData Parameters

The CONNECTDATA data structure (C) and ConnectData class (Java) contain all the information necessary to establish a connection between an external application and Q/LinQ. Both qqSendConnect and qqReceiveConnect use it.

All parameters are character data type, and each field has the same length in C and Java.

Parameter Name	Description	I/O	Req	Length (Bytes)
method	Populated by Q/LinQ. Identifies the type of message being sent or received through the socket. Has the value CONN in this case.	I	N	4
appID	Identifies the application to connect with. If the application initiates the connection to Q/LinQ, this field is not used.	I	Y	12
appVer	Version of the application to which a connection is requested. Can be used for version compatibility checking. If the application is connecting to Q/LinQ, this field is currently not used.	I	N	8
sysID	Identifies the application from which a connection is requested. If Q/LinQ is connecting to the application, this field identifies the host Q/LinQ system. If the application is connecting to Q/LinQ, this field is currently not used.	I	N	10
sysVer	Version of the system from which a connection is requested. Can be used for version compatibility checking. If Q/LinQ is connecting to the application, this field pertains to the host Q/LinQ system. If the application is connecting to Q/LinQ, this field is currently not used.	I	N	8
accessCode	Access code provided to connect with the application, such as a user name or login ID. If the application is connecting to Q/LinQ, this field is currently not used.	I	N	60

Parameter Name	Description	I/O	Req	Length (Bytes)
exImMode	Indicates what type of connection is requested: <ul style="list-style-type: none"> • 1. Export • 0. Import Each connection must be of either one type or the other, never both.	I	Y	1
msgAckLvl	Quality of Service parameter for the level of acknowledgment required for document delivery, used when exporting using middleware. Not currently used when sending documents directly to Q/LinQ.	I	N	8
msgSync	Not implemented. Relevant only to some middleware products. The Quality of Service parameter describing the level of synchronization of an exported document with respect to the destination application. Determines the scope of a Q/LinQ send operation, which can extend to: <ul style="list-style-type: none"> • Only the export of the document to the local transport system. • The delivery of the document to its destination. • The processing of the document by the receiving application. 	I	N	8
msgSecur	Not implemented. Relevant only to some middleware products. The Quality of Service parameter describing the level of security. The security level can be: <ul style="list-style-type: none"> • The authentication required by the destination application to read an exported document. • The level of privacy required. • The type of encryption to be applied to the document data. 	I	N	8
msgPriority	Not implemented. Priority requested for delivery of documents from the application. MOM Quality of Service parameter that overrides middleware settings by application or document.	I	N	8

Parameter Name	Description	I/O	Req	Length (Bytes)
msgRouting	Quality of Service parameter defining routing of document from source to destination, used when exporting using middleware. Not currently used when sending documents directly to Q/LinQ.	I	N	8
msgTTL	Quality of Service parameter for the length of time document is still valid prior to being delivered to the destination. Used when exporting using middleware. Not currently used when sending documents directly to Q/LinQ.	I	N	8
msgClnOrd	Quality of Service parameter indicating whether the order in which a series of documents are exported should determine the order in which they are processed at the destination. Used when exporting using middleware. Not currently used when sending documents directly to Q/LinQ.	I	N	8
msgPersist	Not implemented. Relevant only to some middleware products. The Quality of Service parameter describing the time exported documents should persist inside the communications middleware after Q/LinQ sends them. The persistence time affects recoverability of the message in case of transport failure.	I	N	8
connStatus	Indicates whether or not a valid connection has been made. QQAPI_SUCCESS if connected, otherwise equals the return code for an exception condition.	O	N	4
appHndl	Handle assigned by the listening application to this connection or thread. The connecting application uses it in subsequent messaging API calls when sending or receiving data. If Q/LinQ is requesting a connection with the application, AppHndl is set to session ID by default but may be overwritten with the application's value.	O	N	16

CTRLHEAD and CtrlHead Parameters

The CTRLHEAD data structure (C) and CtrlHead class (Java) contain all the control information about an interface document being sent to or received from Q/LinQ. It is used by `qqSendDocCtrl` and `qqReceiveDocCtrl`. A CTRLHEAD structure or CtrlHead object is required for every Q/LinQ document.

All parameters are character data type, and each field has the same length in C and Java.

Parameter Name	Description	I/O	Req	Length (Bytes)
method	Populated by Q/LinQ. Identifies the type of message being sent or received through the socket. Has the literal value SCTL for a <code>qqSendDocCtrl</code> call and RCTL for a <code>qqReceiveDocCtrl</code> call.	I	N	4
appID	If the interface document is being received from Q/LinQ, this field is the application name as registered in Q/LinQ. If the document is being sent to Q/LinQ, then this field is currently not used.	I	Y	12
appHndl	Handle assigned by the external application to this connection, if necessary. If the interface document is being sent from the application to Q/LinQ, then this field is currently not used.	I	N	16
uoworkType	Indicates how the interface document should be processed along with others created before and after it. Possible values are: <ul style="list-style-type: none"> • 0. Document is independent, relative sequence non-significant. • 1. Document is the first member of a series, where the relative sequence of the document is significant. • 2. Document is the first member of a batch, where all documents in the group must be committed together. • 9. Line is continued from last document. 	I	Y	1

CTRLHEAD and CtrlHead Parameters — (Page 1 of 6)

Parameter Name	Description	I/O	Req	Length (Bytes)
groupID	Identifies the group or unit of work to which the interface document belongs. Not currently used when sending interface documents to Q/LinQ.	I	N	10
groupSeq	Sequence number of the document within its group. Assigned by the external application and not used by Q/LinQ.	I	N	10
docStd	Required for documents that are registered with DocStd values. Published standard that defines the structure and content of the interface document (such as OAGIS).	I	Y	8
docTyp	Required for documents that are registered with document type values. Specific type of interface document or transaction within the standard (such as SYNC_ITEM).	I	Y	21
docRev	Required for documents that are registered with DocRev values. Revision number of the interface document type within its standard (such as 001).	I	Y	8
tradPtrID	Identifies the trading partner from which the interface document originated.	I	N	8
extDocKey	Identifier of the interface document assigned by the external application.	I	N	16
mfgproSite	Identifies the MFG/PRO site to/from which the interface document is directed.	I	N	8
mfgproKey	Identifies any key field or fields relevant to MFG/PRO (such as order-line number, item number) with which the interface document is associated.	I	N	18
destProc	Procedure name to execute in either MFG/PRO or the external application to process the interface document.	I	N	12

CTRLHEAD and CtrlHead Parameters — (Page 2 of 6)

Parameter Name	Description	I/O	Req	Length (Bytes)
ackLvlReqd	Indicates the acknowledgment level requested by the source application, either Q/LinQ or the external application. Valid values are: <ul style="list-style-type: none"> • 0. No confirmation • 1. Confirmation only on failure • 2. Confirmation on success or failure 	I	N	1
origDocId	Identifies an original source document previously exported that is being referenced by the present interface document, expressed in terms of a Q/LinQ-assigned document ID. Passed as a reference from the sending application (such as an acknowledgment).	I	N	8
docLng	Language in which all the text in the interface document is sent, as defined by the ISO language codes.	I	N	2
codePg	Identifies the code page or character set (such as iso8859-1, ibm850) used to present the interface document data.	I	N	15
dateForm	Describes the date format used to represent calendar dates in the interface document, using the values supported by Progress: mdy, dmy, ymd.	I	N	3
numForm	Describes the format used to represent decimal numbers in the interface document, using the values supported by Progress. Possible values are: <ul style="list-style-type: none"> • American. The comma separates thousands and the period is a decimal point. • European. The period separates thousands and the comma is a decimal point. 	I	N	8
dateCreate	Date the document was created by the source application.	I	N	10
timeCreate	Time of day the document was created by the source application.	I	N	10

CTRLHEAD and CtrlHead Parameters — (Page 3 of 6)

Parameter Name	Description	I/O	Req	Length (Bytes)
timeZone	The time zone of DateCreate and TimeCreate , expressed as an offset from Greenwich Mean Time. For example, -0800 for Pacific Time in the US or +0100 for Berlin, Germany.	I	N	5
srcComp	Identifies the source application component that generated the interface document (such as GL or AP), in either MFG/PRO or the external application.	I	N	10
srcTask	Identifies the transaction in the application that caused the interface document to be generated, either MFG/PRO or the external application.	I	N	10
srcProc	Identifies the source procedure in the application that generated the interface document, either MFG/PRO or the external application.	I	N	12
srcUserId	User ID of the person or process initiating the document in either MFG/PRO or the external application, recorded for audit trail purposes.	I	N	8
ctrlTot1	User-defined total accumulated by Q/LinQ for multi-part documents, summed over values provided by MFG/PRO at the time the various lines of the document were published. Can be used for batch counts and reconciling with external applications. Not currently used on documents sent to Q/LinQ.	I	N	5
ctrlTot2	See ctrlTot1 .	I	N	5
ctrlTot3	See ctrlTot1 .	I	N	5
msgAckLvl	Quality of Service parameter for the level of acknowledgment required for interface document delivery. Used by Q/LinQ when exporting using middleware. Not currently used when sending interface documents directly to Q/LinQ.	I	N	8

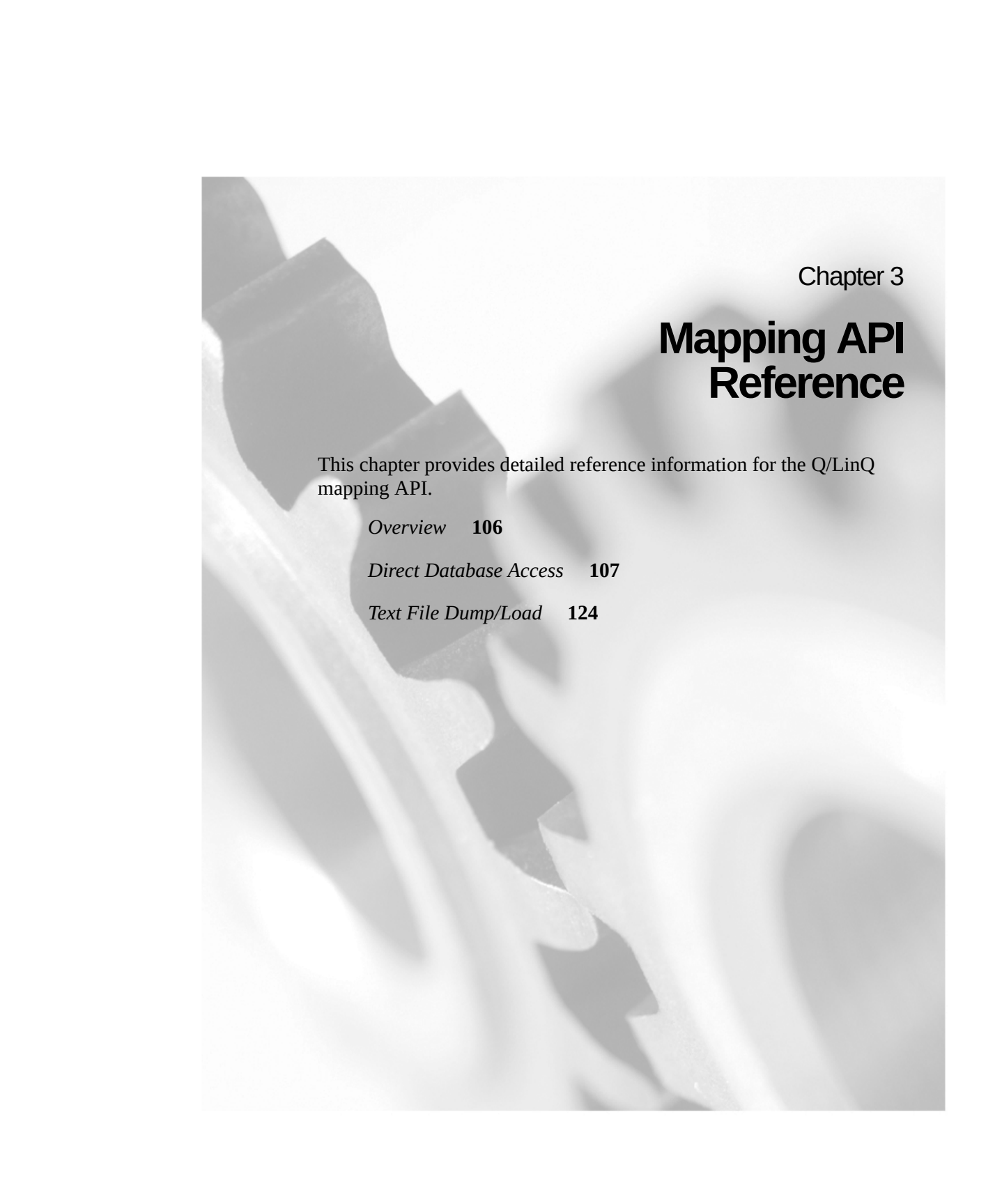
CTRLHEAD and CtrlHead Parameters — (Page 4 of 6)

Parameter Name	Description	I/O	Req	Length (Bytes)
msgSync	<p>Not implemented. Relevant only to some middleware products. The Quality of Service parameter describing the level of synchronization of an exported document with respect to the destination application.</p> <p>Determines the scope of a Q/LinQ send operation, which can extend to:</p> <ul style="list-style-type: none"> • Only the export of the document to the local transport system. • The delivery of the document to its destination. • The processing of the document by the receiving application. 	I	N	8
msgSecur	<p>Not implemented. Relevant only to some middleware products. The Quality of Service parameter describing the level of security.</p> <p>The security level can be:</p> <ul style="list-style-type: none"> • The authentication required by the destination application to read an exported document. • The level of privacy required. • The type of encryption to be applied to the document data. 	I	N	8
msgPriority	Quality of Service parameter for definition of priority level. Used by Q/LinQ when exporting using middleware. Not currently used when sending interface documents directly to Q/LinQ.	I	N	8
msgRouting	Quality of Service parameter defining routing of document from source to destination. Used by Q/LinQ when exporting using middleware. Not currently used when sending interface documents directly to Q/LinQ.	I	N	8

CTRLHEAD and CtrlHead Parameters — (Page 5 of 6)

Parameter Name	Description	I/O	Req	Length (Bytes)
msgTTL	Quality of Service parameter for the length of time document is still valid prior to being delivered to the destination. Used by Q/LinQ when exporting using middleware. Not currently used when sending interface documents directly to Q/LinQ.	I	N	8
msgClnOrd	Quality of Service parameter indicating whether the order in which a series of documents is exported should determine the order in which the documents are processed at the destination. Used by Q/LinQ when exporting using middleware. Not currently used when sending interface documents directly to Q/LinQ.	I	N	8
msgPersist	Not implemented. Relevant only to some middleware products. The Quality of Service parameter describing the time exported documents should persist inside the communications middleware after Q/LinQ sends them. The persistence time affects recoverability of the message in case of transport failure.	I	N	8

CTRLHEAD and CtrlHead Parameters — (Page 6 of 6)

The background of the page is a grayscale photograph of several interlocking mechanical gears. The gears are of different sizes and are arranged in a way that creates a sense of depth and motion. The lighting is soft, highlighting the metallic texture and the sharp edges of the gear teeth.

Chapter 3

Mapping API Reference

This chapter provides detailed reference information for the Q/Linq mapping API.

Overview **106**

Direct Database Access **107**

Text File Dump/Load **124**

Overview

The mapping API invokes procedures for mapping documents between MFG/PRO format and external application formats. The mapping procedure can be:

- A user-written Progress 4GL program.
- A user-written adapter—built using Q/LinQ—to a commercial mapping package. The user-written adapter can use direct database access or text file dump/load.
- Adapter programs supplied with Q/LinQ that work with XSLT stylesheets for mapping OAGIS BODs to native MFG/PRO format.

Note The supplied XSLT stylesheets are built using Extended Stylesheet Language Transformation, the industry standard for XML-based data transformation.

The mapping API calls the mapping procedure for the current document. The default mapping procedure for each document is specified in one of these programs:

- Register External Application (36.8.1.1; 36.8.1 in 9.0)
- Export Specification Maint (36.8.1.2; 36.8.2 in 9.0)
- Import Specification Maint (36.8.1.3; 36.8.3 in 9.0)

The remainder of the chapter addresses the two approaches to building mapping procedures:

- “Direct Database Access” on page 107
- “Text File Dump/Load” on page 124

Direct Database Access

With direct database access, Q/LinQ communicates with a mapping procedure by directly reading and writing the document control tags and data one line at a time from the Q/LinQ queues. This approach is especially useful when writing mapping procedures in Progress, where the mapping logic can perform input and output operations by making Progress 4GL calls without the need for interim text files.

After the mapping procedure program decomposition, this section includes:

- The mapping procedure parameters, page 108
- A mapping procedure code example, page 111
- The mapping procedure routines, from page 113

Program Decomposition: Mapping Procedure

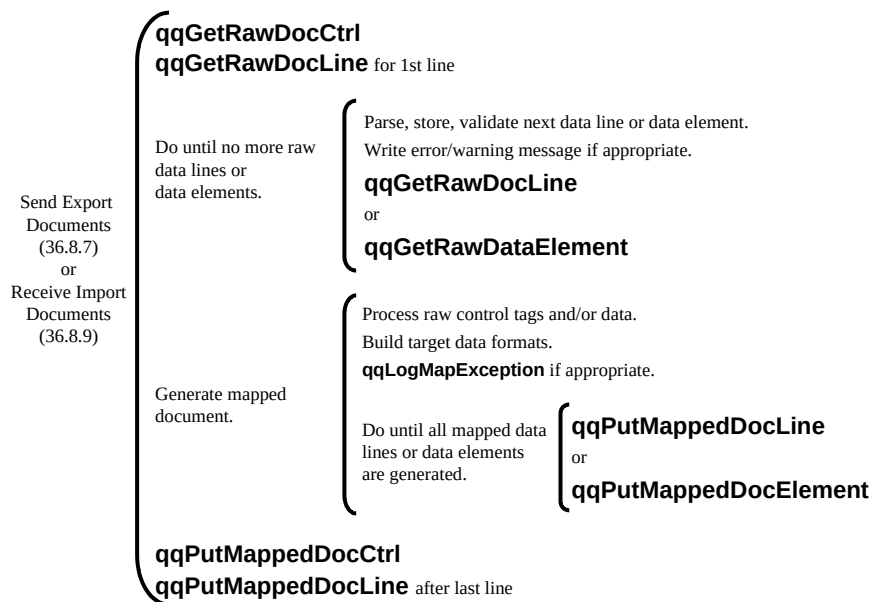
Figure 3.1 shows the structure of a mapping procedure called by the Q/LinQ mapping API. Using its various methods, the procedure:

- Reads the raw Q/LinQ document data one line at a time (`qqGetRawDocLine`) or one field at a time (`qqGetRawDocElement`).
- Writes the mapped data back to Q/LinQ one line at a time (`qqPutMappedDocLine`) or one field at a time (`qqPutMappedDocElement`).
- Reads the document control tags from Q/LinQ with a single call to `qqGetRawDocCtrl`.
- Writes the document control tags back to Q/LinQ after mapping, if necessary, with a call to (`qqPutMappedDocCtrl`).
`qqPutMappedDocCtrl` may be called before or after any calls to `qqPutMappedDocLine` or `qqPutMappedDocElement`, depending on when in the mapping procedure the information to write the document control tags is available.

It is possible to generate multiple mapped documents from a single raw document. To signal the end of one mapped document, call `qqPutMappedDocLine` or `qqPutMappedDocElement` with the `CtrlBreak` parameter set to 2. Subsequent calls to either of these methods

or to `qqPutMappedDocCtrl` causes a new mapped document to be generated with the same document ID as the original but with a different document suffix.

Fig. 3.1
Mapping API
Adapter Logic,
Direct Database
Access



Mapping Procedure Parameters

The entry point of the mapping procedure is the point where it is called by Q/LinQ; in this case, the beginning of the program. The entry point must match the Q/LinQ call signature. That is, it must accept the same parameters of the same type and in the same order as the corresponding call or exit point on the Q/LinQ side.

Table 3.1 lists the input-output parameters that Q/LinQ requires in every mapping procedure called through the mapping API.

Table 3.1
Mapping Procedure
Parameters

Parameter Name	Type	Description	I/O
<code>docId</code>	integer	Document ID of the document being mapped.	I
<code>sessId</code>	integer	The Q/LinQ session calling the mapping procedure.	I

Parameter Name	Type	Description	I/O
exImMode	logical	Yes, if document is an import document; No, if it is an export document.	I
ctrlFileName	x(60)	Path name of the file into which Q/LinQ writes the raw document tags, if the mapping procedure uses the method qqDumpRawDocFiles.	I
dataFileName	x(60)2	Path name of the file into which Q/LinQ writes the raw document tags, if the mapping procedure uses the method qqDumpRawDocFiles.	1
mappedFileName	x(60)	Path name of the file into which the mapping procedure writes any error/warning messages, if the mapping procedure uses the method qqLoadMapExceptionFile.	I
errorFileName	x(60)	Path name of the file into which the mapping procedure writes any error/warning messages, if the mapping procedure uses the method qqLoadMapExceptionFile.	I
mapSpec	x(12)	Map specification ID of the document, for use by the mapping procedure.	I
mapParm1	x(8)	Name of a user-defined parameter to control the mapping procedure.	I
mapParmValue1	x(8)	Value of mapParm1.	I
mapParm2	x(8)	Name of a user-defined parameter to control the mapping procedure.	I
mapParmValue2	x(8)	Value of mapParm2.	I
mapParm3	x(8)	Name of a user-defined parameter to control the mapping procedure.	I
mapParmValue3	x(8)	Value of mapParm3.	I
mapParm4	x(8)	Name of a user-defined parameter to control the mapping procedure.	I
mapParmValue4	x(8)	Value of mapParm4.	I
mapParm5	x(8)	Name of a user-defined parameter to control the mapping procedure.	I
mapParmValue5	x(8)	Value of mapParm5.	I

Parameter Name	Type	Description	I/O
mapParm6	x(8)	Name of a user-defined parameter to control the mapping procedure.	I
mapParmValue6	x(8)	Value of mapParm6.	I
mapParm7	x(8)	Name of a user-defined parameter to control the mapping procedure.	I
mapParmValue7	x(8)	Value of mapParm7.	I
mapParm8	x(8)	Name of a user-defined parameter to control the mapping procedure.	I
mapParmValue8	x(8)	Value of mapParm8.	I
mapParm9	x(8)	Name of a user-defined parameter to control the mapping procedure.	I
mapParmValue9	x(8)	Value of mapParm9.	I
mapParm10	x(8)	Name of a user-defined parameter to control the mapping procedure.	I
mapParmValue10	x(8)	Value of mapParm10.	I
result	integer	<ul style="list-style-type: none"> • 0. Success • <>0. Failure Not necessary to use if <code>qqLogMapException</code> or <code>qqLoadMapExceptionFile</code> methods are used within the mapping procedure.	I
resultTxt	x(160)	Optional text describing any returned error/warning condition. Not necessary to use if <code>qqLogMapException</code> or <code>qqLoadMapExceptionFile</code> methods are used within the mapping procedure.	O
mapperLogPath	x(60)	Optional path name of a file containing a detailed log or trace of all mapping activity for the document.	O
qqHandle	handle	Handle to call mapping API methods.	I

Mapping Procedure Code Example

The following code sample illustrates the structure of a mapping adapter implementing direct database access.

```

/* Entry point for invocation by Q/LinQ */
Define input parameter DocID as integer no-undo.
Define input parameter SessID as integer no-undo.
Define input parameter ExImMode as logical no-undo.
Define input parameter CtrlFileName as character no-undo.
Define input parameter DataFileName as character no-undo.
Define input parameter MappedFileName as character no-undo.
Define input parameter ErrorFileName as character no-undo.
Define input parameter MapSpec as character no-undo.
Define input parameter MapParm1 as character no-undo.
Define input parameter MapParmValue1 as character no-undo.
...
Define input parameter MapParm10 as character no-undo.
Define input parameter MapParmValue10 as character no-undo.
Define output parameter ResultStatus as integer no-undo.
Define output parameter ResultTxt as character no-undo.
Define output parameter MapperLogPath as character no-undo.
Define input parameter QqHandle as handle no-undo.
...
/* Get the raw document control tags from Q/LinQ */
run qqGetRawDocCtrl
    (output DocID, output AppID, output AppVer, output DocStd, output
    DocTyp, output DocRev, output
    TradPtrID, output DestProc, output SysID, output SysVer, output
    SrcComp, output SrcTask,
    output AckLvlReqd, output DocLng, output CodePg, output DateForm,
    output NumForm, output
    SrcUserId, output SrcProc, output MfgproSite, output MfgproKey,
    output ExtDocKey, output
    OrigDocId, output CtrlTot1, output CtrlTot2, output CtrlTot3, output
    DateCreate, output
    NumCreate, output Timezone, output ResultStatus, output ResultTxt).
...

/* Get first raw document line from Q/LinQ */
run qqGetRawDocLine
    (output LineData, output CtrlBreak).
...

/* Repeat until no more lines to read */
...

    /* Parse and validate data line, store for mapping */
    ...

    /* Log exceptions if necessary */
    run qqLogMapException
        (input MsgNbr, input SevLvl, input FldName,
        input Value1, input Value2,
        input RetCd, input MsgTxt).
    ...

```

```

        /* Get next raw document line from Q/LinQ */
run qqGetRawDocLine
    (output LineData, output CtrlBreak).
...
/* End repeat */
/* Repeat until no more lines to map */
...
/* Generate mapped line from parsed input */
...
    /* Write mapped document line */
run qqPutMappedDocLine
    (input LineData, input CtrlBreak,
output ResultStatus).
...
    /* Log exceptions if necessary */
run qqLogMapException
    (input MsgNbr, input SevLvl, input FldName,
    input Value1, input Value2, input Value3,
    input RetCd, input MsgTxt).
...
/* End repeat */
/* Write any mapped document control tags back to Q/LinQ */
run qqPutMappedDocCtrl
    (input DocID, input AppID, input AppVer, input DocStd, input
    DocTyp, input DocRev, input TradPtrID, input DestProc, input
    SrcComp, input SrcTask, input AckLvlReqd, input DocLng, input
    CodePg, input DateForm, input NumForm, input SrcUserId, input
    SrcProc, input MfgproSite, input MfgproKey, input ExtDocKey,
    input OrigDocId, input CtrlTot1, input CtrlTot2, input CtrlTot3,
    input DateCreate, input NumCreate, input Timezone, output
    DocSufx, output ResultStatus).
...

```

qqMappedCode

This routine returns the mapped value of an input code value, where the mappings between discrete source and target values are defined in Code Mapping Maintenance (36.8.1.20). All mappings may be specific to a particular external application ID or document type.

See Chapter 6, “Q/LinQ Utilities,” for detailed information.

Note Code mapping is *not* available in MFG/PRO 9.0.

qqGetRawDocCtrl

This routine retrieves all the document control tags from Q/LinQ for the document to be mapped. While most of these tags are typically not relevant to mapping, others are, such as `DateForm` and `NumForm`.

Use this routine only if you need to see the Q/LinQ document control tags to map the document data.

Parameter Name	Type	Description	I/O
pDocId	integer	Document ID	O
pAppId	char(12)	Application ID	O
pAppVer	char(8)	Application Version	O
pDocStd	char(8)	Document Standard	O
pDocTyp	char(21)	Document Type	O
pDocRev	char(8)	Document Revision	O
pTradPtrId	char(32)	Trading Partner ID	O
pDestProc	char(12)	Destination Procedure	O
pSysId	char(8)	System ID	O
pSysVer	char(8)	System Version	O
pSrcComp	char(10)	Source Component	O
pSrcTask	char(10)	Source Task	O
pAckLvlReqd	char(1)	Acknowledgment Level	O
pDocLng	char(2)	Document Language	O
pCodePg	char(15)	Code Page	O
pDateForm	char(3)	Date Format	O
pNumForm	char(8)	Number Format	O
pSrcUserId	char(8)	Source User ID	O
pSrcProc	char(12)	Source Procedure	O
pMfgproSite	char(8)	MFG/PRO Site	O
pMfgproKey	char(18)	MFG/PRO Key	O
pExtDocKey	char(16)	External Document Key	O
pOrigDocId	char(8)	Original Document ID	O
pCtrlTot1	char(5)	Control Total 1	O
pCtrlTot2	char(5)	Control Total 2	O
pCtrlTot3	char(8)	Control Total 3	O

Parameter Name	Type	Description	I/O
pDateCreate	date	Date Created	O
pTimeCreate	char(8)	Time Created	O
pTimezone	char(5)	Time Zone	O
result	integer	<ul style="list-style-type: none">• 0. Success• 2. Failure	O
resultTxt	char (100)	Text description of the Result parameter (see above)	O

qqGetRawDocLine

This routine retrieves the next sequential line of raw document data from Q/LinQ for the document being mapped. It retrieves multiple lines of the document until reaching the end of the document.

Parameter Name	Type	Description	I/O
lineData	char(1000)	A packet of data from the import document, normally an entire logical line. When there are no more lines remaining, this field is blank.	O
ctrlBreak	char(1)	Control break indicating whether the returned element is: <ul style="list-style-type: none">• 1. At the end of the current document line• 2. At the end of the document after the final document line• 0. Neither	O

qqGetRawDocElement

This routine scans the current line of document data and parses the next triplet from the line, taking into account line continuations that may cause a single triplet to span several physical records.

Parameter Name	Type	Description	I/O	Req
pEOLDelimitOnly	logical	Indicates whether the rest of the current document line following the <code>Context</code> and <code>FieldName</code> tokens, if any, should be considered as the <code>fieldValue</code> .	I	Y
pContext	char	Token of the parsed triplet containing the context.	O	N
pFieldName	char	Token of the parsed triplet containing the field name.	O	N
pFieldValue	char	Token of the parsed triplet containing the field value.	O	N
pCtrlBreak	char	Control break indicating whether the returned element is: <ul style="list-style-type: none"> • 1. At the end of the current document line • 2. At the end of the document after the final document line • 0. Neither 	O	N

qqPutMappedDocCtrl

This routine writes any mapped document control tags to Q/LinQ after mapping is complete. Only control tags modified during the mapping process need to be passed in the program call. While a mapping program typically does not modify most of the tags, some tags might be changed. For example, a mapper might alter the MFG/PRO destination program (DestProc tag) for an imported document based on information extracted from its data.

Use this routine only if you need to change Q/LinQ control tags to map data.

Parameter Name	Type	Description	I/O
pDocId	integer	Document ID	O
pAppId	char(12)	Application ID	I
pAppVer	char(8)	Application Version	I
pDocStd	char(8)	Document Standard	I
pDocTyp	char(21)	Document Type	I
pDocRev	char(8)	Document Revision	I
pTradPtrId	char(32)	Trading Partner ID	I
pDestProc	char(12)	Destination Procedure	I
pSrcComp	char(10)	Source Component	I
pSrcTask	char(10)	Source Task	I
pAckLvlReqd	char(1)	Acknowledgment Level	I
pDocLng	char(2)	Document Language	I
pCodePg	char(15)	Code Page	I
pDateForm	char(3)	Date Format	I
pNumForm	char(8)	Number Format	I
pSrcUserId	char(8)	Source User ID	I
pSrcProc	char(12)	Source Procedure	I
pMfgproSite	char(8)	MFG/PRO Site	I
pMfgproKey	char(18)	MFG/PRO Key	I
pExtDocKey	char(16)	External Document Key	I
pOrigDocId	char(8)	Original Document ID	I
pCtrlTot1	char(5)	Control Total 1	I
pCtrlTot2	char(5)	Control Total 2	I

Parameter Name	Type	Description	I/O
pCtrlTot3	char(5)	Control Total 3	I
pDateCreate	char(8)	Date Created	I
pTimeCreate	char(8)	Time Created	I
pTimezone	char(5)	Time Zone	I
pDocSufx	integer	Mapped document suffix	O
result	integer	<ul style="list-style-type: none">• 0. Success• 2. Failure	O

qqPutMappedDocLine

This routine writes the next line of mapped document data to Q/LinQ for the document being mapped. When called multiple times, it writes multiple lines of the document in ascending order.

Parameter Name	Type	Description	I/O
lineData	char(1000)	A packet of data from the import document, normally an entire logical line. When there are no more lines remaining, this field is blank.	I
ctrlBreak	char(1)	Control break indicating whether the returned element is: <ul style="list-style-type: none">• 1. At the end of the current document line• 2. At the end of the document after the last line has been completed• 0. Neither	I
result	integer	<ul style="list-style-type: none">• 0. Success• 2. Failure	O

qqPutMappedDocElement

This routine appends a single mapped triplet to the end of the current data line being mapped.

Parameter Name	Type	Description	I/O	Req
pContext	char	Context of the mapped document element.	I	N
pFieldName	char	Field name of the mapped document element.	I	N
pFieldValue	char	Field value of the mapped document element.	I	N
pCtrlBreak	char	Control break indicating whether the returned element is: <ul style="list-style-type: none">• 1. At the end of the current document line• 2. At the end of the document after the final document line• 0. Neither	I	N
pResult	integer	<ul style="list-style-type: none">• 0. Success• <>0. Failure	O	N

qqBuildCIMLine

This routine accepts a set of ASCII data values and returns a single string containing all of these values in the CIM format with appropriate delimiters and special characters.

Parameter Name	Type	Description	I/O	Req
pLine	temp-table	Contains a single field value—a character array of extent 50 containing the data values to be mapped into a single line conforming to the CIM format.	I	Y
pNbrFields	integer	Number of fields (up to 50) included in the line.	I	Y
pMaxLen	integer	Maximum line length beyond which the generated CIM line is broken into multiple physical lines. If 0 or >255, 255 characters is assumed.	I	N
pNbrEndKeys	integer	Number of ENDKEY keystrokes that will be appended to the end of the generated CIM line.	I	N
pNbrLines	integer	Number of physical lines making up the generated CIM line.	O	N
pResult	integer	0. Success <>0. Failure	O	N

qqLogMapException

This routine records a data error or warning detected during the mapping process. Input parameters resemble those used by MFG/PRO mfmsg routines.

Parameter Name	Type	Description	I/O
msgNbr	integer	MFG/PRO message number.	I
sevLvl	integer	Message severity level: <ul style="list-style-type: none">• 0. Informational• 1. Warning• 2. Error	I
fldName	char	Name of the field with the error condition.	I
value1	char	The first value included in the MFG/PRO message text.	I
value2	char	The second value included in the MFG/PRO message text.	I
returnCd	char(8)	Return code from the mapper program for the message.	I
resultTxt	char(160)	Text describing the return code, which overrides the MFG/PRO message text.	I

Text File Dump/Load

The text file dump/load approach dumps the raw data and control tags in a file into two files, maps the files, and loads a single mapped file back into Q/LinQ. This approach may be easier to implement than the direct database approach when the mapping program is file oriented (for example, an awk script), but it creates interim text files that increase system overhead.

When an import or export document is associated with a mapping procedure that uses the text file dump/load mapping option, the following steps occur:

- 1 Q/LinQ calls the designated mapping program, which could be a user-written Progress utility.
- 2 The mapping API subroutine `qqDumpRawDocFiles` dumps data from an unmapped document into two files—one containing control tags (`*.qqc`) and the other the raw data (`*.qqr`).
- 3 The mapping program performs the transformation and generates a mapped document file containing modified control tags and the mapped data (`*.qqm`).
- 4 The mapping API subroutine `qqLoadMappedDocFile` loads the files back into Q/LinQ. The `qqLoadMapExceptionFile` subroutine loads the exception file (`*.qqe`) if there is one.

Text File Formats

Q/LinQ requires specific formats for files containing:

- Control tags
- Raw data
- Mapped data
- Exception messages

Control-Tag File Format

Some mapping programs need to access Q/LinQ-maintained control-tag information about a document to properly map it to a target format. Q/LinQ writes the control tags for export and import documents to a text file in the format shown in the following table, with each document attribute, or value, on its own line preceded by a tag name.

Q/LinQ names the file with the session ID and a qqc extension unless the user chooses another name. It ignores tags that are not followed by values. Tag order does not matter and may be changed in future MFG/PRO releases.

When mapping an imported OAGIS BOD, the mapping program is unlikely to require this control-tag file since the BOD is highly self-describing and contains most pertinent document attributes.

Tag	Purpose	Exp	Imp
@DOCID <value>	Q/LinQ-assigned document identifier.	Yes	Yes
@APPID <value>	The source or destination application.	Yes	Yes
@APPVER <value>	The version of the source or destination application that the document came from or goes to.	Yes	Yes
@DOCSTD <value>	Required for documents that are registered with DocStd values. Published standard that defines the structure and content of the document after being mapped by Q/LinQ (such as EDIFACT, EDI X12, CIM, OAGIS BOD).	Yes	Yes
@DOCTYP <value>	Required for documents that are registered with document type values. Specific document type within the document standard after being mapped by Q/LinQ (such as 850 transaction within ANSI X12 standard).	Yes	Yes
@DOCREV <value>	Required for documents that are registered with DocRev values. Revision number of the exported document after being mapped by Q/LinQ.	Yes	Yes
@TRADPTRID <value>	The trading partner that sent the document or receives it.	Yes	Yes

Tag	Purpose	Exp	Imp
@DESTPROC <value>	The destination procedure: <ul style="list-style-type: none"> • If an import document, the MFG/PRO procedure that processes it. It is a Progress 4GL procedure unless the import document is a UI emulation document. Then, the procedure is the MFG/PRO application program whose input data stream Q/LinQ emulates. • If an export document, the procedure at the destination that processes the document. 	Yes	Yes
@SYSID <value>	Logical identifier for Q/LinQ used in interface communications.	Yes	Yes
@SYSVER <value>	Version of the host Q/LinQ system referenced for interface communications. Can be used to verify version compatibility or currency before processing.	Yes	Yes
@DESTPROC <value>	The destination procedure: <ul style="list-style-type: none"> • If an import document, the MFG/PRO procedure that processes it. It is a Progress 4GL procedure unless the import document is a UI emulation document. Then, the procedure is the MFG/PRO application program whose input data stream Q/LinQ emulates. • If an export document, the procedure at the destination that processes the document. 	Yes	Yes
@SRCCOMP <value>	The source application component or module that generated the import or export document; for example, GL or AP.	Yes	Yes
@SRCTASK <value>	The transaction or program within the source application that generated the document; for example, <code>sosomt.p</code> for Sales Order Maintenance in MFG/PRO or Receipt or Adjustment.	Yes	Yes
@ACKLVLREQD <value>	The acknowledgment level the source application requested. Valid values are: <ul style="list-style-type: none"> • 0. No confirmation required • 1. Send message only if not successful • 2. Confirm on success or failure 	Yes	Yes
@DOCLNG <value>	The language that all the text fields in the document were sent in.	Yes	Yes

Tag	Purpose	Exp	Imp
@CODEPG <value>	The code page or character set (such as iso8859-1, ibm850) used to present, or format, the document data.	Yes	Yes
@DATEFORM <value>	Describes the date format used to represent calendar dates in the interface document, using the values supported by Progress: mdy, dmy, ymd.	Yes	Yes
@NUMFORM <value>	Describes the format used to represent decimal numbers in the interface document, using the values supported by Progress. Possible values are: <ul style="list-style-type: none"> American. The comma separates thousands and the period is a decimal point. European. The period separates thousands and the comma is a decimal point. 	Yes	Yes
@SRCUSERID <value>	The user ID of the person or process initiating the export or import of the document.	Yes	Yes
@SRCPROC <value>	The source procedure that created or sent the document.	Yes	Yes
@MFGPROSITE <value>	The primary MFG/PRO host site this Q/LinQ installation is associated with. Use when Q/LinQ is deployed with multi-site MFG/PRO implementations and the site code on MFG/PRO document is populated.	Yes	Yes
@MFGPROKEY <value>	The document identifier containing key reference information (such as order-line number; item number) relevant to MFG/PRO application processing.	Yes	Yes
@EXTDOCKEY <value>	Any value significant to the external application that is useful for audit-trail purposes, such as a cross-reference to an earlier document imported from the application.	Yes	Yes
@ORIGDOCID <value>	The identifier of a document that Q/LinQ passed to the destination application or that the source application passed to Q/LinQ. Used as a reference on an acknowledgement document.	Yes	Yes

Tag	Purpose	Exp	Imp
@CTRLTOT1 <value>	<p>The optional total for multi-part documents:</p> <ul style="list-style-type: none"> For import documents, it is provided by the source application. For export documents, it is a user-defined total accumulated by Q/LinQ and summed over values provided by the source application each time Q/LinQ is called to publish one line of the document. <p>Can be used for batch counts and is useful for reconciling with external applications.</p>	Yes	Yes
@CTRLTOT2 <value>	See CTRLTOT1.	Yes	Yes
@CTRLTOT3 <value>	See CTRLTOT1.	Yes	Yes
@DATECREATE <value>	Date the source application created the import document.		Yes
@TIMECREATE <value>	Time of day the source application created the import document.		Yes
@DATERCVD <value>	Date that Q/LinQ received the import document.		Yes
@TIMERCVD <value>	Time of day that Q/LinQ received the import document.		Yes
@DATEPUBL <value>	Date that Q/LinQ published the export document.	Yes	
@TIMEPUBL <value>	Time of day that Q/LinQ published the export document.	Yes	
@TIMEZONE <value>	For import documents, the time zone of @DATECREATE and @TIMECREATE. For export documents, the time zone setting in Q/LinQ Control.	Yes	Yes

Control-Tag File Format — (Page 4 of 4)

Figure 3.2 shows a sample control-tag file for the exported OAGIS BOD CONFIRM_BOD.

```
@DOCID 673
@APPID lgs-2
@APPVER
@DOCSTD OAGIS
@DOCTYP CONFIRM_BOD
@DOCREV
@TRADPTRID
@DESTPROC
@SYSID MFG/PRO
@SYSVER 9.0
@SRCCOMP Q/LinQ
@SRCTASK Ack
@ACKLVLREQD 0
@DOCLNG us
@CODEPG iso8859-1
@DATEFORM mdy
@NUMFORM AMERICAN
@SRCUSERID rfj
@SRCPROC qqimprc.p
@MFGPROSITE 10000
@MFGPROKEY CustBCT
@EXTDOCKEY referenceid
@ORIGDOCID 305
@CTRLTOT1
@CTRLTOT2
@CTRLTOT3
@DATEPUBL 01/18/1999
@TIMEPUBL 09:28:18
@TIMEZONE +0200
```

Fig. 3.2
Sample Control-
Tag File

Raw-Data File Format

Q/LinQ dumps all the raw data lines of a document to an ASCII file and names it with the session ID and a `qqr` extension unless the user chooses another name. Each line is terminated by a new line (`\n`) character in UNIX or a carriage return/line feed sequence in Windows.

Figure 3.3 shows a sample raw-data file for Q/LinQ to export. Figure 3.4 shows a mapped version of the same data. Figure 3.2 is the control file that wraps the data.

Fig. 3.3Sample Raw-Data
File

```

[CONFIRM]
[CNTRLAREA]
[BSR]
VERB=SYNC
NOUN=CUSTOMER
REVISION=003
[END]
[SENDER]
LOGICALID=lgs-2
COMPONENT=BUSENTITY
TASK=MAINTENANC
REFERENCEID=referenceid
CONFIRMATION=1
LANGUAGE=us
CODEPAGE=64
AUTHID=QAD
[END]
[DATETIME(CREATION)]
DATE=0012/0003/1998
TIME=11:25:11
SUBSECOND=
TIMEZONE=
[END]
[END]
[STATUSLVL]
STATUSLVL=2
[END]
ORIGREF=305
[END]
[CONFIRMSG]
DESCRIPTN=ERROR: PROCEDURE DOES NOT EXIST.
REASONCODE=1
[END]

```

Mapped-Data File Format

For import and export documents, the mapping adapter writes each document control tag name and value on its own line in the mapped file. Q/LinQ requires the mapping program to name the mapped file with the same name as the raw data file and a qqm extension.

Within each document, the tag lines can be in any order, but the tag lines must precede all the data lines. Q/LinQ ignores tag lines that contain no value.

Do not create tags for values that the mapping process does not alter or derive. Q/LinQ changes only those values explicitly appearing in the mapped data file.

When a mapping creates multiple mapped documents from a single raw document (a one-to-many mapping), write the mapped documents to the same mapped-data file, one after the other.

You can override the input- and output-card numbers for the raw-data and mapped-data files, but there is usually no reason to. Override them in the input- and output-card parameter fields (for example, MAPCARD) in:

- Register External Application (36.8.1.1; 36.8.1 in 9.0)
- Export Specification Maint (36.8.1.2; 36.8.2 in 9.0)
- Import Specification Maint (36.8.1.3; 36.8.3 in 9.0)

Figure 3.4 shows the document data of Figure 3.3 in its mapped form. The mapped file can include modified Q/LinQ tags contained in the control file. They go between the @@DOC and @DATA tags. The @DESTPROC tag is a modified tag.

```
@@DOC
@DESTPROC abc.p
@DATA
<?xml version="C2.0" standalone="no"?><!DOCTYPE CONFIRM_BOD_002
SYSTEM
"002_confirm_bod_002.dtd"><CONFIRM_BOD_002><CNTRLAREA><BSR><VERB>
CONFIRM</VERB><NOUN>BOD</NOUN><REVISION>002</REVISION></
BSR><SENDER><LOGICALID>MFG/PRO</LOGICALID><COMPONENT>
Q/LinQ</COMPONENT><TASK>Ack</TASK><REFERENCEID>305</
REFERENCEID><CONFIRMATION>0</CONFIRMATION><LANGUAGE>us</
LANGUAGE><CODEPAGE>iso8859-1</CODEPAGE><AUTHID>rfj</AUTHID></
SENDER><DATETIME qualifier="creation"><YEAR>1999</YEAR><MONTH>01</
MONTH><DAY>18</DAY><HOUR>09</HOUR><MINUTE>28</MINUTE><SECOND>18</
SECOND><SUBSECOND>0000</SUBSECOND><TIMEZONE>+0200</TIMEZONE></
DATETIME></
CNTRLAREA><DATAAREA><CONFIRM_BOD><CONFIRM><CNTRLAREA><BSR><VERB>S
YNC</VERB><NOUN>CUSTOMER</NOUN><REVISION>003</REVISION></
BSR><SENDER><LOGICALID>lgs-2</LOGICALID><COMPONENT>BUSNENTITY</
COMPONENT><TASK>MAINTENANC</TASK><REFERENCEID>referenceid</
REFERENCEID><CONFIRMATION>1</CONFIRMATION><LANGUAGE>us</
LANGUAGE><CODEPAGE>64</CODEPAGE><AUTHID>QAD</AUTHID></
SENDER><DATETIME qualifier="CREATION"><YEAR>1998</
YEAR><MONTH>12</MONTH><DAY>03</DAY><HOUR>11</HOUR><MINUTE>25</
MINUTE><SECOND>11</SECOND><SUBSECOND>0000</
SUBSECOND><TIMEZONE>+0200</TIMEZONE></DATETIME></
CNTRLAREA><STATUSLVL>99</STATUSLVL><ORIGREF>305</ORIGREF></
CONFIRM><CONFIRMMSG><DESCRIPTN>ERROR: PROCEDURE DOES NOT EXIST.</
DESCRIPTN><REASONCODE>1</REASONCODE></CONFIRMMSG></CONFIRM_BOD></
DATAAREA></CONFIRM_BOD_002>
^^ ^^
```

Fig. 3.4
Mapped Data
Format

Message File Format

Q/LinQ logs error messages during mapping and saves them in a file with the extension qqe.

Each line contains detailed information about a single exception message. The message detail is nearly the same information that MFG/PRO applications display from the common mfmsg03.i routines. Each line of the file has the following format:

<message number>,<severity level>,<field name>,
<value 1>,<value 2>,<return code>

Table 3.2 lists each element in the line with a brief description.

Table 3.2
Error File Values

Value	Description
message number	MFG/PRO message number in the msg_mstr table that contains the text of the message.
severity level	1. Informational 2. Warning 3. Error
field name	Name of the field or data element that the exception occurred in. Normally included in the text of an MFG/PRO message to specify the cause of the error or warning.
value 1	An arbitrary string to insert in the text of a generic MFG/PRO message making it more specific to the exception.
value 2	See value 1.
return code	Reference a return code provided by a commercial mapping package that follows an error incident.

The file can list any number of exceptions. When mapping completes, Q/LinQ creates messages in the document log based on the content of the file. The file is set to output card 2 by default, but you can specify the number to use in:

- Register External Application (36.8.1.1; 36.8.1 in 9.0)
- Export Specification Maint (36.8.1.2; 36.8.2 in 9.0)
- Import Specification Maint (36.8.1.3; 36.8.3 in 9.0)

Program Decomposition: Mapping API Adapter

Figure 3.5 shows the general structure of a Progress mapping adapter to map text files.

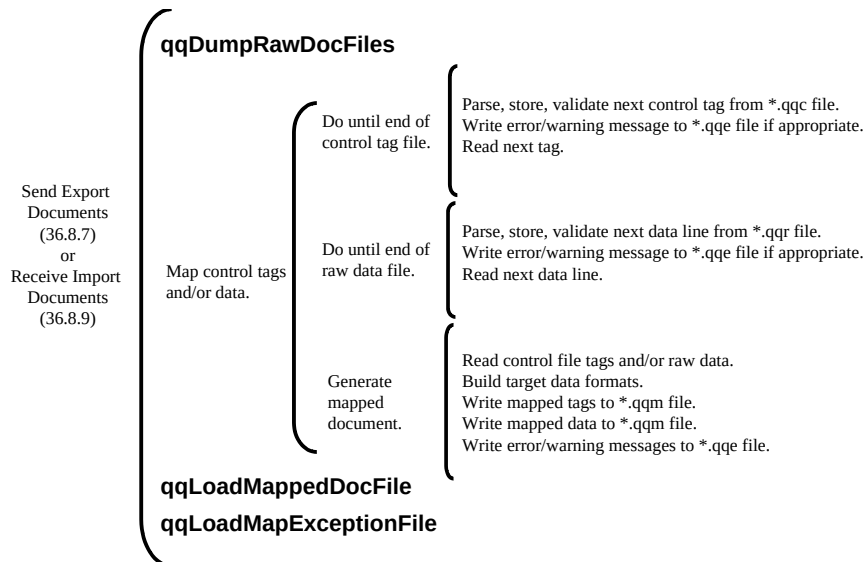


Fig. 3.5
Mapping API
Adapter Logic,
Text File Dump/
Load

The following code sample illustrates an adapter implementing text-file dump/load.

```

/* Entry point for invocation by Q/LinQ */
Define input parameter DocID as integer no-undo.
Define input parameter SessID as integer no-undo.
Define input parameter ExImMode as logical no-undo.
Define input parameter CtrlFileName as character no-undo.
Define input parameter DataFileName as character no-undo.
Define input parameter MappedFileName as character no-undo.
Define input parameter ErrorFileName as character no-undo.
Define input parameter MapSpec as character no-undo.
Define input parameter MapParm1 as character no-undo.
Define input parameter MapParmValue1 as character no-undo.
...
Define input parameter MapParm10 as character no-undo.
Define input parameter MapParmValue10 as character no-undo.
Define output parameter ResultStatus as integer no-undo.
Define output parameter ResultTxt as character no-undo.
Define output parameter MapperLogPath as character no-undo.

```

```

...
/* Dump the raw document control tags and data to ASCII files prior
to mapping */
run qqDumpRawDocFiles
    (input-output CtrlFileName, input-output DataFileName, output
    ResultStatus).

/* Marshall mapping parameters and call mapping program to parse, map
data; and write results to mapped data, exception files */
...

/* Load the mapped document data into Q/LinQ after mapping completed
*/
run qqLoadMappedDocFile
    (input-output MappedFileName, output ResultStatus, output
    MappedDocCount).

/* Load the exception data, if any, into Q/LinQ */
run qqLoadMapExceptionFile
    (input-output ErrorFileName, output ResultStatus).

```

qqDumpRawDocFiles

This routine dumps raw control tags and data into two text files with Q/LinQ-specific tag format.

Parameter Name	Type	Description	I/O
ctrlFileName	char(12)	Name of the file where raw I/O document tags are written.	I/O
dataFileName	char(12)	Name of file where raw document data are written.	I/O
result	integer	<ul style="list-style-type: none">• 0. Success• 2. Failure	O

qqLoadMappedDocFile

This routine loads a text file into Q/LinQ. The file contains mapped document data and control tags in a Q/LinQ-specific format.

Parameter Name	Type	Description	I/O
dataFileName	char(12)	Name of file that raw document data resides in.	I/O
result	integer	<ul style="list-style-type: none">• 0. Success• 2. Failure	O
mappedDocCount	integer	Number of mapped documents generated from the raw document.	O

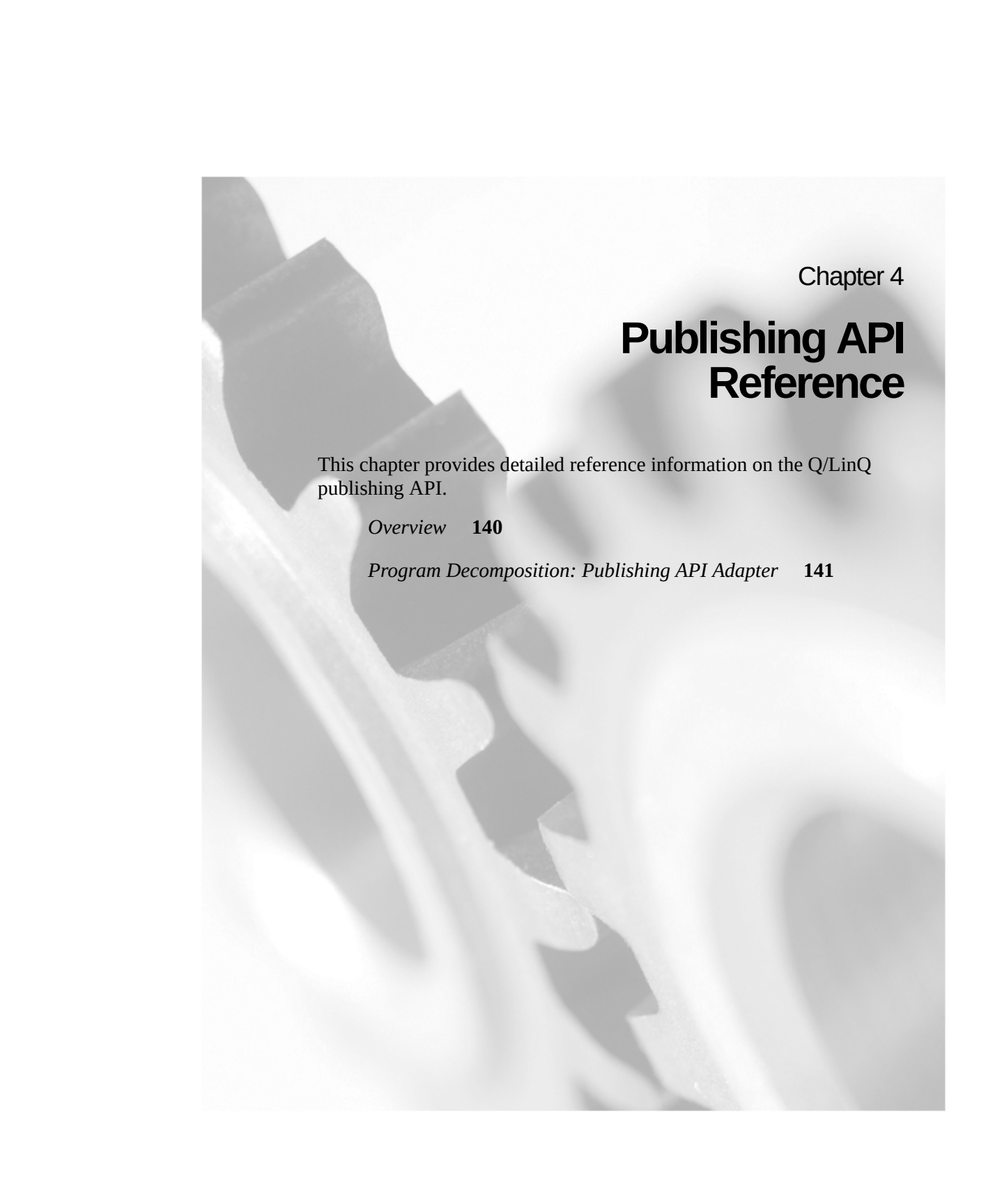
qqLoadMapExceptionFile

This routine loads into Q/LinQ a text file containing exception messages generated by the mapping process. This file must have the Q/LinQ-specific format.

Parameter Name	Type	Description	I/O
errFileName	char(12)	Name of file to load that contains error and warning messages.	I/O
result	integer	<ul style="list-style-type: none">• 0. Success• 2. Failure	O

The input and output parameters for qqLogMapException, qqGetRawDocCtrl, and qqGetRawDocLine are the same as the ones used for direct database access.

▶ See “Direct Database Access” on page 107.



Chapter 4

Publishing API Reference

This chapter provides detailed reference information on the Q/LinQ publishing API.

Overview **140**

Program Decomposition: Publishing API Adapter **141**

Overview

The Q/LinQ publishing API queues documents from MFG/PRO in Q/LinQ for export to an external application or file.

▶ See page 149 for the triplet format.

When publishing data, the Q/LinQ publishing adapter, `qqpubfwd.p`, uses either of two formats:

- An ASCII-based syntax, called the triplet format, similar to that used in most `.ini` files
- XML, especially the QAD-defined QDoc format

Note XML format is not supported in MFG/PRO 9.0.

▶ See *Technical Reference: QXtend Inbound* for details.

QDocs deliver inbound data to MFG/PRO menu-level programs in a proprietary XML format. They are generated with a tool called QGen included with QXtend Inbound, a component of the QAD QXtend interoperability framework.

▶ QXtend Inbound is not available on MFG/PRO 9.0.

The framework provides a single, standardized data interface between QAD products, and between QAD products and external systems. The interface is a Web services-based, SOAP-compliant, XML framework, enabling complete platform-independent access to MFG/PRO business functionality.

Any program written in Progress—including MFG/PRO programs—can be customized to use the publishing adapter. The custom Progress program calls the publishing adapter to pass the documents it assembled to Q/LinQ for export. Q/LinQ handles the actual output operations.

The publishing adapter uses the same internal procedures for publishing as it does for forwarding. A passed parameter (`DESTAPPID`) indicates whether a document is for publishing or forwarding.

Note Forwarding is exporting documents from one MFG/PRO session and placing them in the import queue for processing by another MFG/PRO session in the same database.

Program Decomposition: Publishing API Adapter

All the publishing and forwarding routines are internal procedures of the persistent procedure `qqpubfwd.p` accessible by adapter programs. Adapters can also use the `qqMappedCode` function to map data values.

Q/LinQ maintains raw and mapped data images in its export queue. `qqpubfwd.p` places raw data in this queue for later publishing or forwarding.

Triplet Format

The following illustrates the general structure of a publishing adapter that generates documents in the triplet format:

```
/*Set the document type and other control information*/
/*Populate the input parameters for qqBeginInterfaceDoc*/
/*Call internal procedure to create document log*/
...
    {gprunp.i "qqpubfwd" "p" "qqBeginInterfaceDoc" "()"}
...
/*Repeat until all the document data has been included in */
/*the document*/
...
/*Determine or derive a single field or line of */
/*document data and populate the input parameters for */
/*qqBuildDocElement*/
/*Call internal procedure to create a triplet of raw or */
/*mapped data*/
...
    if return-value = 0 then
        {gprunp.i "qqpubfwd" "p" "qqBuildDocElement"
         "()"}
...
/*End repeat*/
...
/*Publish the export document to Q/LinQ*/
    if return-value = 0 then
        {gprunp.i "qqpubfwd" "p" "qqPublishInterfaceDoc"
         "()"}
...

```

◆ See parameter list on page 145.

◆ See parameter list on page 152.

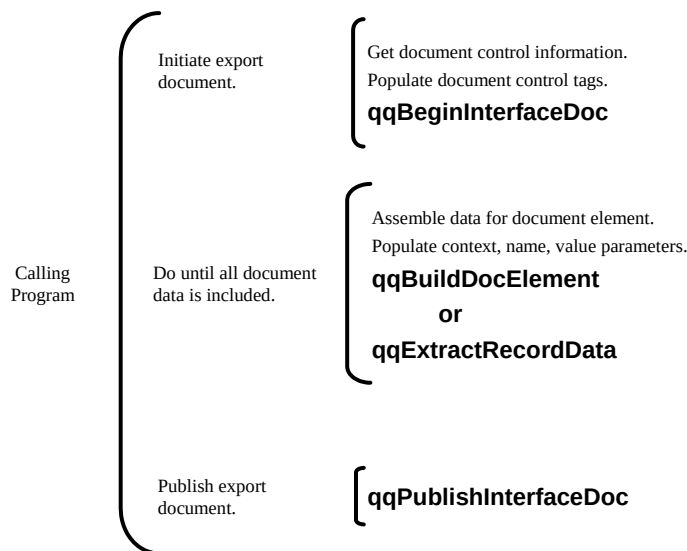
◆ See parameter list on page 155.

If `qqPublishHndl` returns a value of 0, the publishing API executed successfully. Any other value indicates an error occurred.

When the error is corrected, run `qqPublishInterfaceDoc` again and set its `CancelFlg` parameter to Yes (see page 155). This reinitializes all the temporary work tables created by the program without publishing any data. Then rerun the program from the beginning.

Figure 4.1 illustrates the logic of the publishing in triplet format.

Fig. 4.1
Publishing API
Adapter Logic for
Triplet Documents



XML Format

The following illustrates the general structure of a publishing adapter that generates QDoc XML documents. Unlike the triplet case, use of the publishing API consists of a single call to `qqPublishXMLDoc`, but requires that the XML document be built beforehand using the XML capabilities of the Progress 4GL. It accepts an X-DOCUMENT handle to an XML document, and serializes it into the Q/LinQ export queue.

Details on DOM Builder are in *Technical Reference: QXtend Inbound*.

Outside of Q/LinQ, MFG/PRO contains a set of XML Document Object Model (DOM) Builder procedures that automate many of the low-level details involved in building an XML document containing MFG/PRO data. In particular, it can structure the resulting XML document as a

QDoc that conforms to basic QDoc syntax rules. Use of the DOM Builder procedures is not required to publish XML documents to Q/LinQ, but is highly recommended.

```

/*
 * Use Progress XML DOM object support to build your XML document.
 * If possible, call the DOM Builder methods to create the XML data
 * and store it in a handle. The key methods are createNewXMLDocument,
 * createNewQdoc, addNodeGroup, addRecordNode, addNode,
 * addAttributes, getXMLAsTempTable.
 */

{pxrun.i
  &PROC = createNewXMLDocument
  &PARAM = "(
    ...)"
  &HANDLE = domBuilder
  &CATCHERROR = true

{pxrun.i
  &PROC = addNode
  &PARAM = "(
    ...)"
  &HANDLE = domBuilder
  &CATCHERROR = true
}

{pxrun.i
  &PROC = addNodeGroup
  &PARAM = "(
    ...)"
  &HANDLE = domBuilder
  &CATCHERROR = true
}

...

{pxrun.i
  &PROC = getXMLAsTempTable
  &PARAM = "(input (buffer exportDoc:handle))"
  ...
  &HANDLE = domBuilder
  &CATCHERROR = true
}

/*Set the document type and other control information*/
/*Populate the input parameters for qqPublishXMLDoc*/
/*Call internal procedure to create document log*/
...

{gprunp.i "qqpubfwd" "p" "qqBeginInterfaceDoc" "(<parameters>)"
{pxrun.i
  &PROC = qqPublishXMLDoc
  &PARAM = "(
    ...)"
  &HANDLE = domBuilder
  &CATCHERROR = true
}

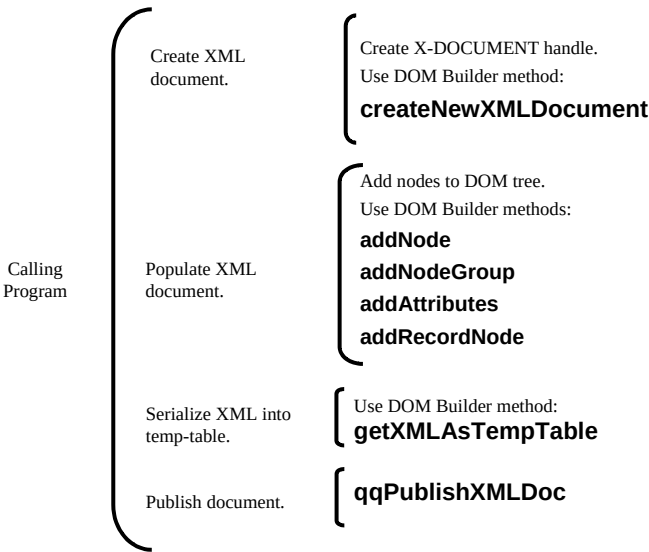
```

See parameter list
on page 145.

If the `qqDocPublished` output parameter contains a value of `true`, the publishing API call executed successfully. Otherwise, an error occurred and the XML document could not be published.

Figure 4.2 illustrates the logic of publishing in XML format.

Fig. 4.2
Publishing API
Adapter Logic for
XML Documents



qqBeginInterfaceDoc

The procedure `qqBeginInterfaceDoc` initiates the creation of an export document in Q/LinQ. It returns a new document ID and, based on the value of the destination application ID, creates either an export or import document log record for the new document. It also returns the next unused group ID number.

Q/LinQ uses the group ID to determine the processing sequence for documents.

Once `qqBeginInterfaceDoc` is called, it cannot be called again from within the same log-in session until the initiated document is published, forwarded, or canceled using the `CancelFlg` parameter of `qqPublishInterfaceDoc`. Each active export document requires its own MFG/PRO log-in session.

`qqBeginInterfaceDoc` first verifies that the destination application exists in the Q/LinQ registry and then checks whether the application is enabled for publishing.

The following table lists parameters of `qqBeginInterfaceDoc`.

Parameter Name	Type	Description	I/O	Req
<code>destAppId</code>	x(12)	Identifies the Q/LinQ registered application to which the document is to be exported. If the System ID from Q/LinQ Control is entered here, then the document is forwarded as an import back into Q/LinQ.	I	Y
<code>docStd</code>	x(8)	Required for documents that are registered with <code>DocStd</code> values. Published standard that defines the structure and content of the document after being mapped by Q/LinQ (such as EDIFACT, EDI X12, CIM, OAGIS BOD).	I	Y
<code>docTyp</code>	x(21)	Required for all documents. Specific document type within the document standard after being mapped by Q/LinQ (such as 850 transaction within ANSI X12 standard).	I	Y

Parameter Name	Type	Description	I/O	Req
docRev	x(8)	Required for documents that are registered with docRev values. Revision number of the exported document with respect to its document standard.	I	Y
unitOfWork	x(1)	Indicates how a group of documents exported together should be treated. Possible values are: <ul style="list-style-type: none"> • 0. No document sequencing or grouping • 1. Documents sequenced but processed independently • 2. Documents sequenced and processed as a group If the previous document published by this session used unitOfWork and was not lastOfGroup, then its unit of work is continued and this parameter is ignored.	I	Y
lastOfGroup	logical	When unitOfWork is in effect, a true value indicates that this document is the last element of the specified unit of work.	I	Y
docId	integer	Uniquely identifies the generated document for Q/LinQ.	O	Y
groupID	integer	Number used to group document for sequenced processing. Returned only if unitOfWork is being used and no Group ID is currently open for this log-in session.	O	Y
tradPtrID	x(32)	Identifies the external party to which the document should be sent.	I	N
mfgproSite	x(8)	Identifies the MFG/PRO site from which the document is being sent. Can be used during multiple-database processing to indicate the source database of a document.	I	N
mfgproKey	x(18)	Any value significant to the MFG/PRO user in referencing the document to relevant MFG/PRO application activity (such as order-line number, item number). Can be used to generate reports from Q/LinQ and locate documents.	I	N

Parameter Name	Type	Description	I/O	Req
extDocKey	x(16)	Any value significant to the external application that may be useful for audit-trail purposes, such as a cross-reference back to an earlier document imported from the application.	I	N
origDocID	integer	Identifies an earlier Q/LinQ document that was referenced by the destination application. Used when process acknowledgment is implemented.	I	N
ackLvlReqd	x(1)	Indicates what acknowledgment level was requested: <ul style="list-style-type: none"> • 0. No confirmation required. • 1. Send message only if not successful. • 2. Confirm on success or failure. 	I	N
destProc	x(12)	Procedure to be executed at destination to process the document.	I	N
docLng	x(2)	Language in which all the text fields in the document were sent, as defined by the ISO language codes.	I	N
srcComp	x(10)	Identifies the source application component that generated the document (such as GL, AP).	I	N
srcTask	x(10)	The transaction or program within the source application that generated the document; for example, <code>SOSOMT . p</code> for Sales Order Maintenance in MFG/PRO or Receipt or Adjustment.	I	N
srcProc	x(12)	The source procedure that created or sent the document.	I	N
srcUserID	x(8)	The user ID of the person or process initiating the export or import of the document.	I	N
useQueue	logical	Not implemented. Indicates whether published documents should be queued by Q/LinQ prior to import or export.	I	N
accessCode	x(60)	Access code that lets Q/LinQ communicate with the application.	I	N
appVer	x(8)	The application version of the document's source or destination.	I	N

Parameter Name	Type	Description	I/O	Req
emailUserID	x(8)	The MFG/PRO user ID of the person to notify by e-mail of successful processing or errors or warnings Q/LinQ encountered during processing of the documents.	I	N
emailNotifyLvl	x(1)	Valid values are: <ul style="list-style-type: none"> • 0. No e-mail for any event. • 1. Q/LinQ encountered errors during processing. • 2. Q/LinQ encountered errors or warnings. • 3. Notify of errors, warnings, and successful processing. 	I	N

qqBeginInterfaceDoc — (Page 4 of 4)

qqBuildDocElement

The procedure `qqBuildDocElement` adds one element to the document being published. It accepts up to three related data values. This set of values is called a triplet. The procedure formats them in an ASCII text string with delimiters, or separator characters. With each successive `qqBuildDocElement` call, more data appends to the end of the document. Q/LinQ stores the document in a temporary table for later processing by `qqPublishInterfaceDoc`.

To create a triplet of raw data, provide the data to Q/LinQ in the form of tagged data fields using the `qqBuildDocElement` parameters `Context`, `FldName`, `FldValue`, and `LineBreak`.

Each time the procedure is called, it publishes a single triplet with the following format:

```
[Context]FldName=FldValue
```

- `Context`, enclosed in square brackets, distinguishes a field name that occurs in different contexts or segments within the same document, such as different lines of a multiline journal entry. It can also identify the data type and segment in OAGIS BODs. If `Context` has no value, the brackets are omitted from the output.
- `FldName` is a user-defined name corresponding to a field in the database schema, program variable name, or other data element meaningful to the destination application. The procedure appends an equal sign to the `FldName` value. If `FldName` has no value, the equal sign is omitted from the output.
- `FldValue` is the value of the data to store.
- Use `LineBreak` between each document element unless you want to generate a delimiter-separated-value data stream. The format is useful for exporting to programs such as Microsoft Excel but it is not suitable for more complex, structured data such as OAGIS BODs.

If `LineBreak` is absent in MFG/PRO 9.0, Q/LinQ inserts a pipe character (|) as the delimiter between elements. In the other versions, Q/LinQ inserts the export delimiter character defined in Register External Application (36.8.1.1; 36.8.1 in 9.0) or Export Specification Maint (36.8.1.2; 36.8.2 in 9.0).

The `Context`, `FldName`, and `FldValue` parameters are optional and not validated by Q/LinQ. They let the programmers writing the adapters make interface data self-describing and easy to parse. If the `qqBuildDocElement` call omits `Context` and `FldName`, `qqBuildDocElement` needs only the `FldValue` parameter. It applies no delimiters to the data, which the calling program must have converted to character data type before passing it to `qqBuildDocElement`.

Example An item number within a published document might be:

```
pt_part=01-10000
```

The same item ordered on line 3 of a sales order could be:

```
[sod_det]sod_line =3|sod_part=01-10000
```

In terms of OAGIS field names, the same item number within a published document might be:

```
ITEMNO=01-10000
```

The same item ordered on line 3 of a sales order, using OAGIS field names might be:

```
[SOLINE]
LINENO=3
ITEMNO=01-10000
...
[END]
```

Using the `Context`, `FldName`, and `FldValue` parameters, the formatting conventions of `qqBuildDocElement` support the generation of several common data formats such as:

- .ini files:

```
[context]
FldName=FldValue
FldName=FldValue
FldName=FldValue
...
[END]
```

- Delimiter-separated values:

```
FldValue|FldValue|FldValue...
```

- Delimiter-separated tagged values:

```
FldName=FldValue|FldName=FldValue|...
```

When publishing MFG/PRO data that will be mapped into OAGIS BODs, use the `.ini` file approach. Begin each data type and segment within the BOD with a unique `Context` value and end each with a `Context` of `END`. This gives the structures a well-defined nesting.

▶ See Figure 3.3 on page 130.

If the data type, segment, and field names are OAGIS-defined names, they can directly map to the corresponding extensible markup language (XML) tags defined in the BOD's data type definition (DTD).

When the procedure's `LineBreak` parameter is `Yes`, it publishes the triplet that follows it on a new line. If it is `No`, a delimiter character separates the triplet from the next one.

Note In MFG/PRO 9.0, use `LineBreak` to separate data triplets only if they contain pipe characters that must be read literally. In the other versions, use `LineBreak` to separate data triplets only if there is no defined delimiter or if the delimiter must be read literally.

Use the optional parameters `CtrlTotalEntry1`, `CtrlTotalEntry2`, and `CtrlTotalEntry3` to store accumulated control totals for multi-part documents, such as line counts or hash totals.

Example To count the number of logical lines loaded into the export queue, set `CtrlTotalEntry1` to 1 when running `qqBuildDocElement` for the last element of a logical line. For all other elements, set `CtrlTotalEntry1` to the Progress undefined value (?). When `qqPublishInterfaceDoc` is called, it accumulates all the values in each temp-table record and stores them in the export or import document log.

Optional parameters can supply the Progress undefined value (?) to the program instead of a value.

An export document must have been initiated using `qqBeginInterfaceDoc` before `qqBuildDocElement` can be used.

The following table lists parameters of `qqBuildDocElement`.

Parameter Name	Type	Description	I/O	Req
<code>context</code>	x(12)	Used to distinguish among multiple field names that are within the same segment of a document. This parameter would not normally be used when the document does not require mapping. When publishing OAGIS BODs, <code>Context</code> should be set to the data type name that it is a member of, such as <code>SOHEADER</code> or <code>SOLINE</code> in the <code>SYNC_SALESORDER</code> BOD.	I	Y
<code>fldName</code>	x(8)	A fully qualified name corresponding to a field in the schema or a program variable. This parameter would not normally be used when the document does not require mapping. It must be used if <code>context</code> is also used. For OAGIS BODs, <code>fldName</code> should be set to the field name of the data inside the BOD; for example, <code>SALESORDERID</code> .	I	Y
<code>fldValue</code>	x(1000)	Data value of the field identified in <code>FldName</code> . Q/LinQ accepts field values of arbitrary length.	I	Y
<code>lineBreak</code>	logical	If Yes, break line here and start a new document line for the data that follows. If No (default), append a pipe character () in Version 9.0 or a user-registered delimiter character in other versions to the published line and continue.	I	Y
<code>ctrlTotalEntry1</code>	integer	User-defined entry for control totals accumulated by Q/LinQ.	I	N
<code>ctrlTotalEntry2</code>	decimal	User-defined entry for control totals accumulated by Q/LinQ.	I	N
<code>ctrlTotalEntry3</code>	decimal	User-defined entry for control totals accumulated by Q/LinQ.	I	N

qqExtractRecordData

The procedure `qqExtractRecordData` is an alternative to the `qqBuildDocElement` procedure available to:

- Q/LinQ installations that also use the MFG/PRO Data Synchronization product available for MFG/PRO 9.0, eB, and eB2
- Any Q/LinQ installation on MFG/PRO eB2.1

Note Data synchronization functions are included with Q/LinQ beginning with MFG/PRO eB2.1.

It extracts the values of a predefined set of data elements from a single MFG/PRO database record, provided the record meets predefined selection criteria. The data elements and selection criteria are defined as part of a synchronization profile using Synchronization Profile Maintenance (36.8.22.1) on the Data Synchronization Setup menu.

Assuming that the synchronization profile and database record are known to the calling program, a single call to `qqExtractRecordData` has the same effect as many successive calls to `qqBuildDocElement`, serializing the names and values of the designated MFG/PRO table and fields into Q/LinQ triplet strings. The triplet strings are placed into a temporary table for later processing by `qqPublishInterfaceDoc`.

The segments of each triplet are populated by `qqExtractRecordData` as follows. There are no line breaks inserted between the triplets.

- `Context` contains the name of the MFG/PRO table (for example, `pt_mstr`) from which the data is being extracted. It appears only once at the beginning of the triplet string and is omitted for each field.
- `FldName` is set to the name of the MFG/PRO field (for example, `pt_part`), and is provided for every extracted data element. There is also an `action` field published, whose value is `A` (for add).
- `FldValue` is the value of the MFG/PRO data element.

Example Given a synchronization profile defined for the `pt_mstr` table containing only the fields `pt_part`, `pt_desc`, `pt_um`, and `pt_prod_line`, a triplet line created by `qqExtractRecordData` might appear as follows:

```
[pt_mstr]|action=A|pt_part=01-10-10000|pt_desc=0ASIS(TM)
COOLING SYSTEM HOME/INDUST MODEL|pt_um=EA|pt_prod_line=
1000
```

An export document must have been initiated using `qqBeginInterfaceDoc` before `qqExtractRecordData` can be used.

The following table lists the parameters of `qqExtractRecordData`.

Parameter Name	Type	Description	I/O	Req
<code>tableName</code>	char	Name of the MFG/PRO table containing the designated record to be published.	I	Y
<code>extractRowId</code>	rowid	ROWID of the buffer containing the database record with data to be extracted.	I	Y
<code>profileId</code>	char	Synchronization profile ID defining the data to be extracted.	I	Y
<code>result</code>	integer	Indicates if the requested data was extracted. Valid values are: <ul style="list-style-type: none">• 0. Success• <> 0. Failure	O	Y

qqPublishInterfaceDoc

The procedure `qqPublishInterfaceDoc` signals Q/LinQ that all the data for the export document has arrived and is ready for publishing. The procedure loops through the records created by `qqBuildDocElement` and packs them into either raw or mapped document records for export or import. After the call to `qqPublishInterfaceDoc`, use `qqBeginInterfaceDoc` to initiate another document.

`qqPublishInterfaceDoc` provides a parameter for canceling a document that has not yet been published to the Q/LinQ log or queues. This option may be useful in cases where it is not known whether the document should be published until after all the data has been assembled.

The following table lists parameters of `qqPublishInterfaceDoc`.

Parameter Name	Type	Description	I/O	Req
<code>cancelFlg</code>	logical	Yes cancels the publish operation by reinitializing the temporary work tables.	I	N
<code>ctrlTotal1</code>	integer	Control total 1 accumulated by Q/LinQ.	O	N
<code>ctrlTotal2</code>	decimal	Control total 2 accumulated by Q/LinQ.	O	N
<code>ctrlTotal3</code>	decimal	Control total 3 accumulated by Q/LinQ.	O	N

qqPublishXMLDoc

Tip

This procedure is not available in MFG/PRO 9.0.

The procedure `qqPublishXMLDoc` initiates the creation of an export document in Q/LinQ, accepts all the data from a temp-table, and publishes the document. It returns a new document ID and, based on the value of the destination application ID, creates either an export or import document log record for the new document. It also returns the group ID number of the document. It performs roughly the same work for XML documents that the two methods `qqBeginInterfaceDoc` and `qqPublishInterfaceDoc` perform for triplet documents.

Q/LinQ uses the group ID to determine the processing sequence for documents.

`qqPublishXMLDoc` should not be called until the XML document has been built using the Progress 4GL language features supporting XML DOM objects, and serialized into a temp-table. It is called only once for a given document.

▶ See *Technical Reference: QXtend Inbound* for details on DOM Builder.

Outside of Q/LinQ, MFG/PRO contains a set of XML DOM Builder procedures that automate many of the low-level details involved in building an XML document containing MFG/PRO data. In particular, it can structure the resulting XML document as a QDoc that conforms to basic QDoc syntax rules. Use of the DOM Builder procedures is not required to publish XML documents to Q/LinQ, but is highly recommended.

The following table lists parameters of `qqPublishXMLDoc`.

Parameter Name	Type	Description	I/O	Req
<code>docStd</code>	x(8)	Required for documents that are registered with <code>DocStd</code> values. Published standard that defines the structure and content of the document after being mapped by Q/LinQ (such as EDIFACT, EDI X12, CIM, OAGIS BOD, qdoc).	I	Y
<code>docTyp</code>	x(21)	Required for all documents. Specific document type within the document standard after being mapped by Q/LinQ (such as 850 transaction within ANSI X12 standard).	I	Y

qqPublishXMLDoc — (Page 1 of 3)

Parameter Name	Type	Description	I/O	Req
docRev	x(8)	Required for documents that are registered with DocRev values. Revision number of the exported document with respect to its document standard.	I	Y
destAppId	x(12)	Identifies the Q/LinQ registered application to which the document is to be exported. If the System ID from Q/LinQ Control is entered here, then the document is forwarded as an import back into Q/LinQ.	I	Y
tradPtrID	x(32)	Identifies the external party to which the document should be sent.	I	N
profileId	x(16)	Not yet implemented.		
mfgproSite	x(8)	Identifies the MFG/PRO site from which the document is being sent. Can be used during multiple-database processing to indicate the source database of a document.	I	N
mfgproKey	x(18)	Any value significant to the MFG/PRO user in referencing the document to relevant MFG/PRO application activity (such as order-line number, item number). Can be used to generate reports from Q/LinQ and locate documents.	I	N
extDocKey	x(16)	Any value significant to the external application that may be useful for audit-trail purposes, such as a cross-reference back to an earlier document imported from the application.	I	N
origDocID	integer	Identifies an earlier Q/LinQ document that was referenced by the destination application. Used when process acknowledgment is implemented.	I	N
ackLvlReqd	x(1)	Indicates what acknowledgment level was requested: <ul style="list-style-type: none"> • 0. No confirmation required. • 1. Send message only if not successful. • 2. Confirm on success or failure. 	I	N
qqPublishXMLDoc — (Page 2 of 3)				

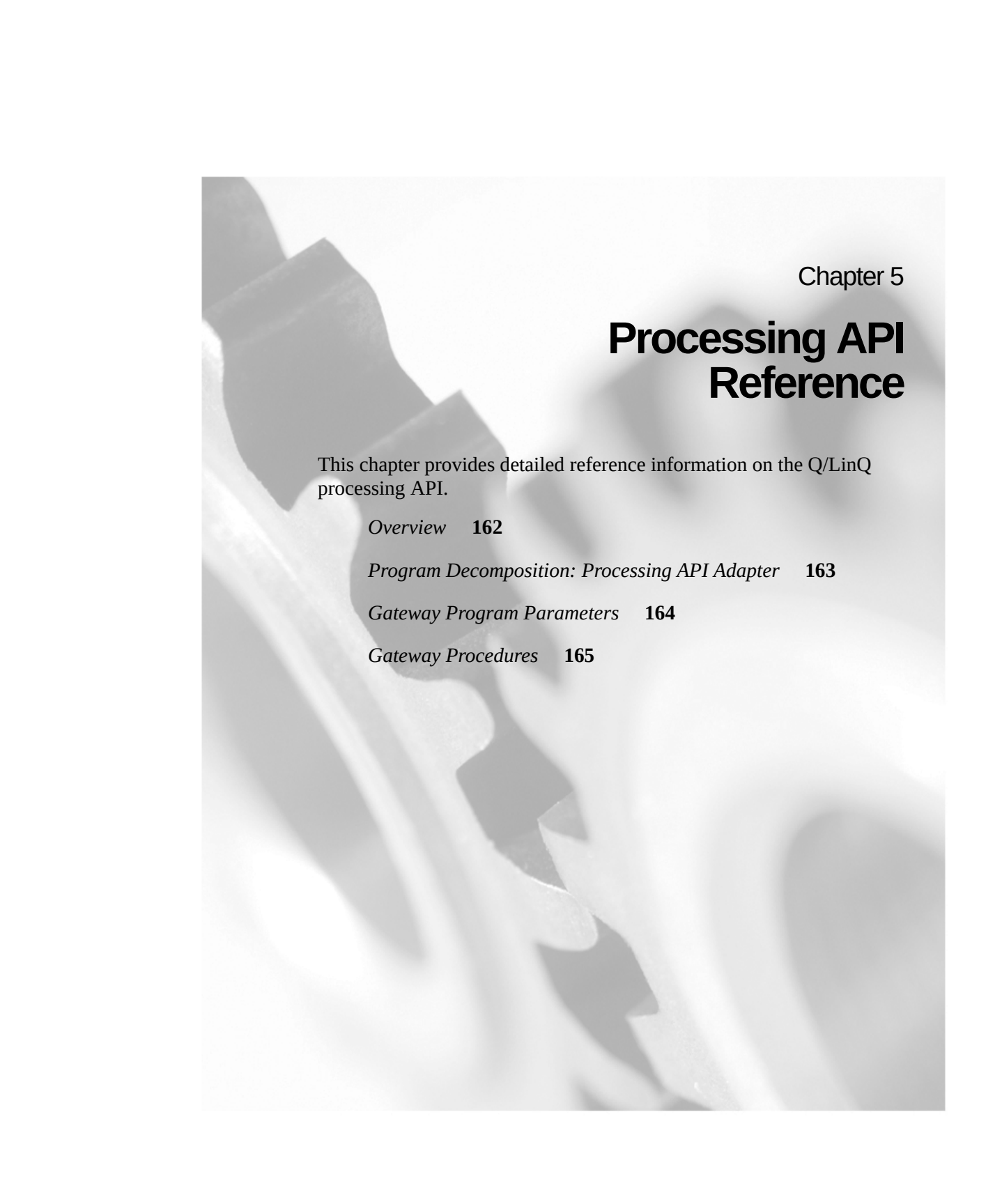
Parameter Name	Type	Description	I/O	Req
unitOfWorkType	x(1)	<p>Indicates how a group of documents exported together should be treated. Possible values are:</p> <ul style="list-style-type: none"> • 0. No document sequencing or grouping. • 1. Document sequenced but processed independently. • 2. Group member: Document sequenced and processed in a single unit of work with the preceding and/or following document if their unit of work types are also 2. • 9. Last of group: Document sequenced and processed in a single unit of work with the preceding document if its unit of work type is 2. <p>If the previous document published by this session used <code>unitOfWork</code> and was not <code>lastOfGroup</code>, then its unit of work is continued and this parameter is ignored.</p>	I	Y
doNotQueue	logical	Not yet implemented.	I	Y
xmlDocument	handle	Handle of the <code>ttXMLDocument</code> temp-table containing the serialized XML document. This temp-table is defined in <code>gpdmcrl.i</code> and is populated by the <code>getXMLAsTempTable</code> method of the MFG/PRO DOM Builder procedures.	I	Y
docPublished	logical	Indicates if the document was successfully published by Q/LinQ.	O	Y
docId	integer	Uniquely identifies the generated document for Q/LinQ.	O	Y
groupId	integer	Q/LinQ-assigned number used to group document for sequenced processing.	O	Y
<i>qqPublishXMLDoc</i> — (Page 3 of 3)				

qqMappedCode

This function returns the mapped value of an input code value, where the mappings between discrete source and target values are defined in Code Mapping Maintenance (36.8.1.20). All mappings may be specific to a particular external application ID or document type.

See Chapter 6, “Q/LinQ Utilities,” for detailed information.

Note Code mapping is not available in MFG/PRO 9.0.

The background of the page is a grayscale image of several interlocking gears. The gears are of different sizes and are arranged in a way that creates a sense of depth and mechanical complexity. The lighting is soft, highlighting the metallic texture of the gears.

Chapter 5

Processing API Reference

This chapter provides detailed reference information on the Q/LinQ processing API.

Overview **162**

Program Decomposition: Processing API Adapter **163**

Gateway Program Parameters **164**

Gateway Procedures **165**

Overview

The processing API either emulates UI data entry or directly loads the MFG/PRO database using a gateway program.

This chapter only discusses loading with gateway programs. UI emulation is covered in *External Interface Guide: Q/LinQ*.

Gateway programs are user-written Progress procedures. They have no UI. Instead, they directly load data from documents into the MFG/PRO database.

When using a gateway program, set the Process Through User Interface field to No in Register External Application (36.8.1.1; 36.8.1 in 9.0) and Import Specification Maint (36.8.1.3; 36.8.3 in 9.0).

Process Import Documents (36.8.10) calls a gateway program when:

- It encounters a document type that has been associated with a gateway program.
- It is importing from an external application associated with the gateway program.

Q/LinQ then passes control information to the gateway through input parameters rather than by emulating a UI.

The entry point of the gateway program is the point where it is called by Q/LinQ—the beginning of the program. It must match the call signature; that is, use input and output parameters of the same type and in the same order as the corresponding call or exit point on the Q/LinQ side.

Program Decomposition: Processing API Adapter

Figure 5.1 shows the processing API and gateway program structure. The gateway wrapper is a custom program that includes calls to processing API procedures.

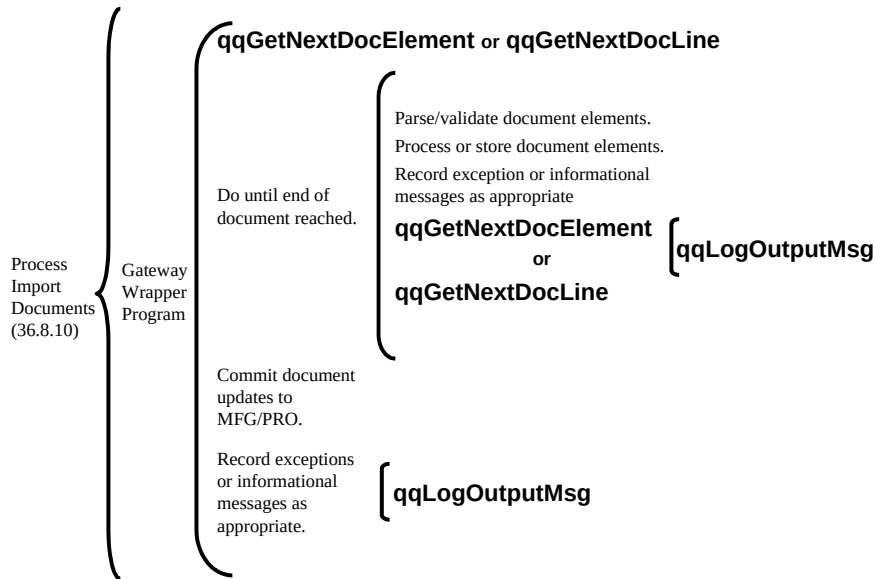


Fig. 5.1
Processing API
Gateway Program
Logic

The following sample shows the general structure of a gateway program implementing the processing API:

```

/*Entry point for invocation by Q/LinQ*/
/*Get first document element from Q/LinQ*/
  run qqGetNextDocElement in CallingProc (output EOLDelimitOnly,
  output Context, output FieldName, output FieldValue, output
  CtrlBreak, output ResultStatus).
  ...
  /*Repeat until no more document elements*/
  if CtrlBreak="2" then leave
  ...
/*Process document element into MFG/PRO*/
...
/*Log error and warning messages,if any, into Q/LinQ*/
  run qqLogProcException in CallingProc
  (input MsgNbr, input SevLvl, input Var1, input Var2, input Var3).
  ...

```

```

/*Get next document element from Q/LinQ*/
  run qqGetNextDocElement in CallingProc (output EOLDelimitOnly,
    output Context, output FieldName, output FieldValue, output
    CtrlBreak, output ResultStatus).
...
/*End repeat*/

```

Gateway Program Parameters

The following table lists the input-output parameters that Q/LinQ requires in every gateway program called through the processing API.

Parameter Name	Type	Description	I/O	Req
appID	x(12)	Application that sent the import documents.	I	Y
DocStd	x(8)	Required for documents that are registered with DocStd values. Published standard that defines the structure and content of the document after being mapped by Q/LinQ (such as EDIFACT, EDI X12, CIM, OAGIS BOD).	I	Y
docTyp	x(21)	Required for documents that are registered with docType values. Specific document type within the document standard after being mapped by Q/LinQ (such as 850 transaction within ANSI X12 standard).	I	Y
sessID	integer	The Q/LinQ session that calls the gateway program.	I	Y
qqHandle	handle	Handle to call processing API methods.	I	Y
result	x(1)	Indicates the success or failure of the processing. Valid values are: <ul style="list-style-type: none"> • 0. Successful • 1. Warnings only • 2. Errors returnStat result goes into the document's Q/LinQ log.	O	Y
docRev	x(8)	Revision number of the imported document after being mapped by Q/LinQ.	I	N
tradPtrID	x(32)	Identifies the trading partner that sent the document.	I	N

Parameter Name	Type	Description	I/O	Req
mfgproSite	x(8)	Identifies the MFG/PRO site that the document is being sent to. Can be used during multiple-database processing to indicate the source database of a document.	I	N
mfgproKey	x(18)	Any value significant to the MFG/PRO user in referencing the document to relevant MFG/PRO application activity (such as order-line number, item number). Can be used to generate reports from Q/LinQ and locate documents.	I	N
extDocKey	x(16)	Any value significant to the external application that may be useful for audit-trail purposes, such as a cross-reference back to an earlier document imported from the application.	I	N

Gateway Procedures

The gateway wrapper program makes use of three methods of the Q/LinQ processing API program qqimprc.p:

- qqGetNextDocElement
- qqGetNextDocLine
- qqLogProcException

Note qqLogProcException replaces the qqLogOutputMsg method, which has been deprecated.

The procedures are accessed as callback methods by using the CallingProc parameter passed into the gateway program as a procedure function handle, as shown in the code example on page 163.

This program can also use qqMappedcode to map data values.

qqGetNextDocElement

This routine retrieves from Q/LinQ the next document element of the imported document. It reads the data field by field or element by element; the gateway program does not have to parse the significant data out of each line.

Each document element is assumed to be expressed using the Q/LinQ triplet format, stored as the three tokens `Context`, `FieldName`, and `FieldValue` as follows:

```
[<Context>]<FieldName>=<FieldValue>
```

Any of the tokens can be omitted, and multiple elements within a document can be placed either on the same line separated by delimiter characters or on separate lines. In MFG/PRO 9.0, the pipe character (|) is always used as the delimiter character. In other versions, the delimiter character is user-defined as the Import Delimiter in Import Specification Maint (36.8.1.3; 36.8.3 in 9.0) or Register External Application (36.8.1.1; 36.8.1 in 9.0). This is the same format used by Q/LinQ to publish export data.

The following table lists parameters of `qqGetNextDocElement`.

Parameter Name	Type	Description	I/O	Req
eOLDelimitOnly	logical	Indicates how the returned document element field value terminates: <ul style="list-style-type: none"> • Yes. Terminates only at the end of the current document line • No. Also terminated by a delimiter character, either pipe () for 9.0 or a user-registered character in other versions Setting this field to Yes lets the routine parse data values that contain the pipe () character.	I	N
context	x(32)	Context (first token) of the next document element or triplet. Can be null if the element contains no context token.	O	N
fieldName	x(32)	Field name (second token) of the next document element or triplet. Can be null if the element contains no field name token.	O	N

Parameter Name	Type	Description	I/O	Req
fieldValue	x(1000)	Field value (third token) of the next document element or triplet. Can be null if the element contains no field value token. Can include delimiter characters only if <code>eOLDeLimitOnly</code> is Yes.	O	N
ctrlBreak	x(1)	Control break indicating whether the returned element is: <ul style="list-style-type: none"> • 1. The end of the current document line • 2. The end of the document, after the final line has been processed • 0. Neither If <code>eOLDeLimitOnly</code> is Yes, it always signals the end of either the current line or the document.	O	N
resultStatus	integer	Indicates if the data returned is not an entire line due to buffer-size limitations. If Yes, the next call retrieves the next packet of data within the same line.	O	N

qqGetNextDocLine

This routine retrieves the next line of the import document from Q/LinQ. It is analogous to a read or import statement of the next line of an ASCII file. It requires no input parameters because the calling Q/LinQ program tracks which document is being processed and which lines have been read.

Use this procedure instead of qqGetNextDocElement to have the gateway program process an entire unparsed line of data at one time. This is necessary whenever the import document is mapped to a format other than the Q/LinQ triplet format. For import documents mapped to the triplet format, use qqGetNextDocElement.

You can use qqGetNextDocElement and qqGetNextDocLine together to process both element by element and line by line through different sections of a single document. However, this practice is not recommended. If qqGetNextDocLine is called after qqGetNextDocElement, but the latter has not read the entire contents of its current line, Q/LinQ jumps to the next line and the remaining content of the first line is not provided to the gateway program.

The following table lists parameters of qqGetNextDocLine.

Parameter Name	Type	Description	I/O	Req
lineData	x(1000)	A packet of data from the import document, normally an entire logical line. When there are no more lines remaining, this field is blank.	O	Y
ctrlBreak	x(1)	Control break indicating whether the returned element is: <ul style="list-style-type: none"> • 1. At the end of the current document line • 2. The end of the document, after the final line has been processed • 0. Neither If eOLDelimitOnly is Yes, it always signals the end of either the current line or the document.	O	Y

qqLogProcException

This routine sends an output message to Q/LinQ. Q/LinQ saves the message in the document's log, along with a reference to an MFG/PRO or external key value if desired. The output messages are typically errors or warnings of data exceptions. If the output is an error message, Q/LinQ records the document status as Error. View the document log record detail in a report. Q/LinQ can automatically notify designated MFG/PRO users of errors through e-mail messages.

This method replaces the qqLogOutputMsg method, which has been deprecated.

The following table lists parameters of qqLogProcException.

Parameter Name	Type	Description	I/O	Req
msgNbr	integer	MFG/PRO message number describing the exception condition.	I	Y
sevLvl	integer	Indicates the severity level of the message. Valid values are: <ul style="list-style-type: none"> • 0. Informational • 1. Warning • 2. Error 	I	Y
var1	x(8)	Variable 1 for inclusion in the body of the MFG/PRO message, depending on the message format.	I	N
var2	x(8)	See var1.	I	N
var3	x(8)	See var1.	I	N
extDocKey	x(16)	Reference significant to the external application that is the source of the processed document.	I	No
mfgProKey	x(18)	MFG/PRO key value associated with the exception, to help Q/LinQ users locate it and diagnose its cause.	I	No

qqLogOutputMsg

See page 169.

Important The qqLogOutputMsg method has been deprecated and replaced by qqLogProcException. qqLogOutputMsg may be removed from a future Q/LinQ release and should no longer be used.

This routine sends an output message to Q/LinQ. Q/LinQ saves the message in the document's log. The output messages are typically errors or warnings of data exceptions. If the output is an error message, Q/LinQ records the document status as Error. View the document log record detail in a report. Q/LinQ can automatically notify designated MFG/PRO users of errors through e-mail messages.

The following table lists parameters of qqLogOutputMsg.

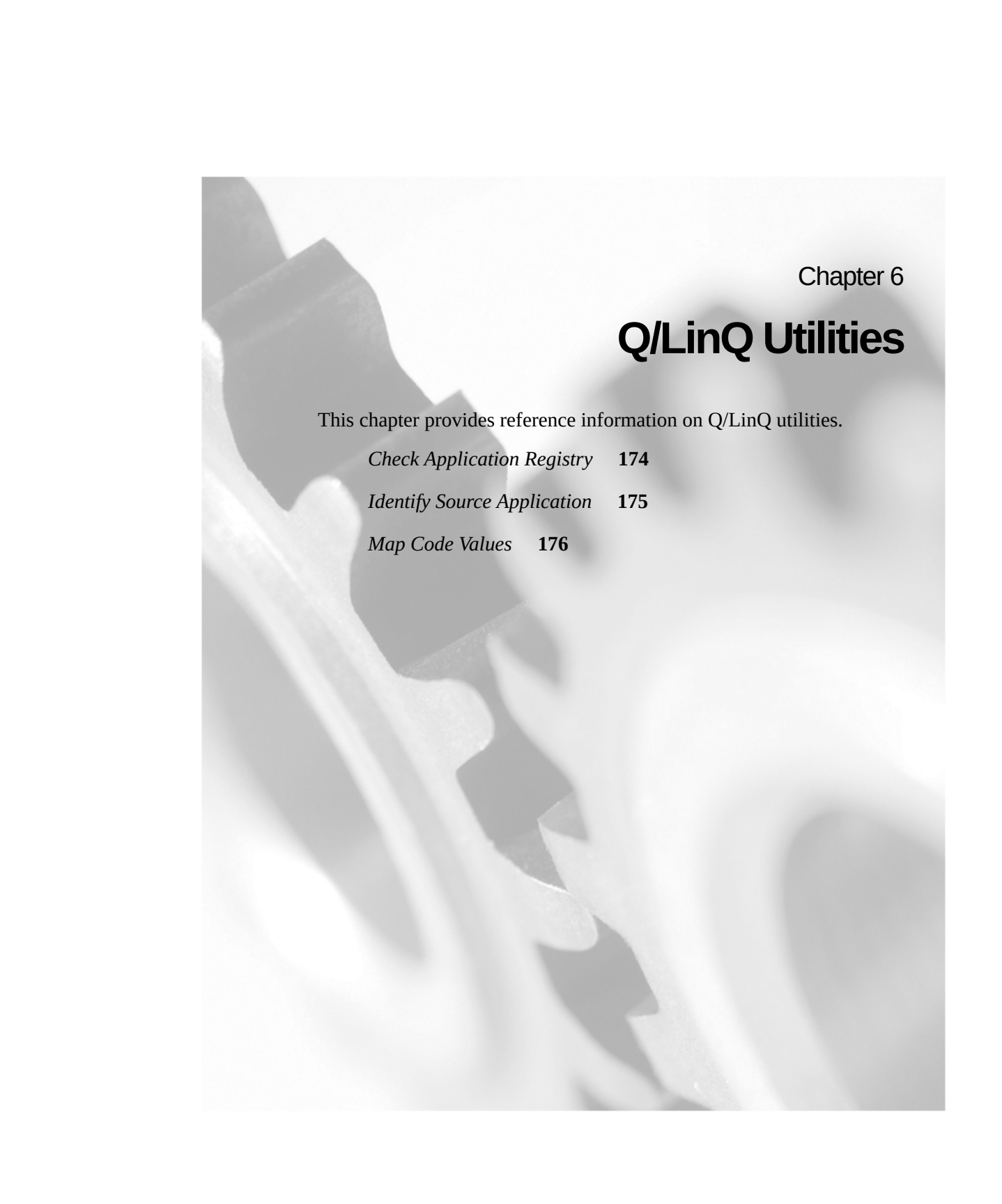
Parameter Name	Type	Description	I/O	Req
msgNbr	integer	MFG/PRO message number describing the exception condition.	I	Y
sevLvl	integer	Indicates the severity level of the message. Valid values are: <ul style="list-style-type: none"> • 0. Informational • 1. Warning • 2. Error 	I	Y
var1	x(8)	Variable 1 for inclusion in the body of the MFG/PRO message, depending on the message format.	I	N
var2	x(8)	See var1.	I	N
var3	x(8)	See var1.	I	N

qqMappedCode

This function returns the mapped value of an input code value, where the mappings between discrete source and target values are defined in Code Mapping Maintenance (36.8.1.20). All mappings may be specific to a particular external application ID or document type.

See Chapter 6, “Q/LinQ Utilities,” for detailed information.

Note Code mapping is not available in MFG/PRO 9.0.

The background of the page is a grayscale photograph of several interlocking mechanical gears. The gears are of different sizes and are positioned in a way that creates a sense of depth and motion. The lighting is soft, highlighting the metallic texture and the sharp edges of the gear teeth.

Chapter 6

Q/LinQ Utilities

This chapter provides reference information on Q/LinQ utilities.

Check Application Registry **174**

Identify Source Application **175**

Map Code Values **176**

Check Application Registry

Use `mgckapp.p` to:

- Determine whether an external application has been registered in Register External Application (36.8.1.1; 36.8.1 in 9.0)
- Retrieve basic information about the external application

Call this subroutine to check entries in the Q/LinQ registration table without directly accessing it. Using this subroutine insulates the adapter from later Q/LinQ schema changes when implementing adapters in the MFG/PRO environment that make validations against entries in Register External Application.

Parameter Name	Type	Description	I/O	Req
appID	x(8)	Identifies the external application to Q/LinQ.	I	Y
appExists	logical	Indicates whether the requested application is registered in Q/LinQ.	O	Y
publFlg	logical	Indicates whether the application is enabled for publishing documents.	O	Y
exMapFlg	logical	Indicates whether the application is enabled for mapping exported documents.	O	Y
sendFlg	logical	Indicates whether the application is enabled for sending documents.	O	Y
rcvFlg	logical	Indicates whether the application is enabled for receiving documents.	O	Y
imMapFlg	logical	Indicates whether the application is enabled for mapping imported documents.	O	Y
procFlg	logical	Indicates whether the application is enabled for processing imported documents.	O	Y
appDesc	x(60)	Description of the application.	O	N
appName	x(20)	Name of the application.	O	N
appVer	x(8)	Version of the application.	O	N

Identify Source Application

Use `mgqqapp.i` while processing MFG/PRO transactions to determine:

- Whether the current transaction is an imported Q/LinQ document
- The source application

Call this subroutine from a user-written program while it processes transactions.

This subroutine is particularly useful when processing documents through standard online processing programs using Q/LinQ's user-interface emulation. It does not affect the MFG/PRO program's processing, but can be used to store the identifier of the source application that owns the data.

After obtaining the identification, you can store an imported document's source application ID with the document data in the MFG/PRO database and associate the document with it.

Parameter Name	Type	Description	I/O	Req
ownerField	character	Field name in which to store the source application ID of the current transaction. The value is the identity of the imported document's source application, which is defined in Register External Application. If the current transaction was not imported using Q/LinQ, the field is blank, and the routine has no effect.	I	Y

Example of use:

```
{mgqqapp.i "pt_mstr.pt_app_owner"}
```

Map Code Values

qqMappedCode returns the mapped value of an input code value where the mappings between discrete source and target values are defined in Code Mapping Maintenance (36.8.1.20).

Note qqMappedCode is not available in MFG/PRO Version 9.0.

Use qqMappedCode to:

- Translate the source value of a particular MFG/PRO code into a target value understood by an external application.
- Translate the source value of a particular code received from an external application into a target value understood by MFG/PRO.

Parameter Name	Type	Description	I/O	Req
fieldName	x(32)	Name of the field to be mapped	I	Y
sourceValue	x(32)	Value of the field to be mapped	I	Y
<return value>	x(32)	Mapped value of the field	O	

This function supports only discrete, one-to-one code translations that do not require the use of formulas, operators, or other functions. All input and return values are expressed in ASCII character strings; no other character types are supported. The source and target values for the mapping are defined in Code Mapping Maintenance and reported by Code Mapping Inquiry (36.8.1.21).

The mapped code value returned by qqMappedCode is determined not only by the source value provided as input, but also by the application ID, document standard, document type, document revision, and/or trading partner ID of the Q/LinQ document being mapped, published, or processed at the time qqMappedCode is used.

Code Mapping Maintenance allows the source and target values to be defined based on a progressively more specific set of search keys, where the most specific search key is used first. If no target value is available for the given source value using any portion of the search key, the source value is returned as the target value.

Example The following code mappings could be established in Code Mapping Maintenance:

Application ID	Document Type	Field Name	Source Value	Target Value
Pur-East	PO-RECEIPT	FREIGHT-TYP	A	1
Pur-East	SO-SHIP	FREIGHT-TYP	A	2
Pur-West	PO-RECEIPT		A	3

- When processing a PO-RECEIPT document from the Pur-East application, the expression `qqMappedCode ("FREIGHT - TYP", "A")` would return 1.
- When publishing a SO-SHIP document to the Pur-East application, `qqMappedCode ("FREIGHT - TYP", "A")` would return 2.
- When mapping a PO-RECEIPT document received from the Pur-West application, which has no code mappings defined, `qqMappedCode ("FREIGHT - TYP", "A")` would return A.

Note that defining the source and target code values in Code Mapping Maintenance is not sufficient to enable its use. The user-written program utilizing the Q/LinQ mapping, publishing, or processing API must invoke `qqMappedCode( )`, supplying the input parameters in all those cases where it will be used. The function must also be defined by inserting the following code at the beginning of the user-written program that will invoke it:

```
{qqmpcdef.i}
```


Glossary

Adapter. A program that mediates differences in data communications or remote method-invocation protocols. An adapter translates call signatures or parameters of one program into a form usable by another—the adapted program. This lets the two programs exchange data or lets the calling program invoke the services of the adapted program without having to modify code. Do not confuse with a data-mapper program, which mediates differences in application data content.

ADI. See *Application Data Interface (ADI)*.

ANSI X12. American National Standards Institute, Standard X12.

Apache Software Foundation. A membership-based, not-for-profit corporation that exists to provide organizational, legal, and financial support for the Apache open-source software projects.

API. See *Application Program Interface (API)*.

Application Data Interface (ADI). A data-transfer capability in MFG/PRO programs using the Object-Based Component Model (OBCM) architecture. OBCM allows data streams going into or out of MFG/PRO to be redirected from or to an external text file. Unlike CIM Interface records, the data format of ADI records is independent of the user

interface presentation. It includes metadata consisting of Progress-qualified field names as well as data values, providing improved flexibility and maintainability.

Application Program Interface (API). A set of entry points into and calls out of an application. External applications use these to link to and pass data or provide services to an application without having to modify its internal processing. Commercial APIs are typically exposed to external programs through designated program libraries with the signature of each call, the call parameters, and the procedures for use well documented.

Asynchronous Processing. Processing that an external event (for example, committing a transaction to the database) initiates but is not completed as part of the same unit of work that initiated it. Asynchronous processing can complete long after the originating event completes. The originating application is not blocked from further processing before the asynchronous processing finishes.

Batch of One Immediate (BOOI). The preparation of transactions for immediate transport between applications as an application event occurs, as opposed to subsequent or periodic extraction and batching of business data for later transport. BOOI data is not necessarily synchronously received or processed at the destination. Implementations

often employ a store-and-forward architecture to first queue the data, then send it later based on data communications availability, cost, and timing requirements at the destination.

BOD. See *Business Object Document (BOD)*.

BOOI. See *Batch of One Immediate (BOOI)*.

Business Object Document (BOD). The OAGIS implementation of the concept of document (see *Document*). Business object documents specify the business data that are exchanged between different kinds of applications.

Callback Function. An API entry point that permits external programs to invoke services of an application through a direct program call into that application. References to such entry points are often passed to the external applications through subscribe or connect functions.

Common Object Request Broker

Architecture (CORBA). An architecture and specification for creating, distributing, and managing distributed program objects in a network. Programs at different locations and developed by different vendors communicate in a network through an interface broker. See also *Object Request Broker (ORB)*.

Document. General term used to signify an instance of a business transaction, such as a sales order or invoice. In Q/LinQ, a document is an electronic representation of business data sent to or received from an external application or trading partner. This term is generally used at the application level of abstraction, in preference to the somewhat more technical terms transaction or message. See also *Mapped Document* and *Raw Document*.

Document Object Model (DOM).

A programming interface specification developed by the W3C that lets a programmer create and modify HTML pages and XML documents as full-fledged program objects. As program objects, the data in structured documents can be hidden, ensuring control over who can manipulate the document, and the documents can carry with them predefined methods.

EDIFACT. Electronic Data Interchange for Administration, Commerce, and Transport. Generally used to refer to the set of international standards that were developed to enable EDI among businesses located in North America, Europe, and other geographic regions. In 1987, the EDIFACT syntax proposals were accepted by the ISO Technical Board and the number ISO 9735 was allotted. These standards are application, communications medium, and machine-independent.

Electronic Data Interchange (EDI). The electronic transport of business documents (such as sales, purchases, shipment notices, delivery schedules) between trading partners. Various industries and QAD target markets, such as automotive manufacturing and consumer-products distribution, use EDI extensively. These industries generally implement EDI using commercial enabling software products that support data transport, often using value-added-network (VAN) services, and data translation to or from the industry-defined EDI standard formats.

Extensible Markup Language (XML). A flexible way to create information formats and share both the format and the data on the Web, intranets, and elsewhere. XML is similar to HTML; both contain markup symbols to

describe the contents of a page or file. HTML, however, describes only how a Web page displays. XML describes its content and data types.

Extensible Stylesheet Language (XSL). A language for formatting an XML document; for example, showing how the data described in the XML document should be presented in a Web page. XSLT shows how the XML document should be reorganized into another data structure (which could then be presented by following an XSL style sheet).

Extensible Stylesheet Language Transformation (XSLT). A standard way to describe how to transform the structure of an XML document into an XML document with a different structure. The coding for the XSLT is also referred to as a style sheet and can be combined with an XSL style sheet or be used independently.

External System. A set of programs running on a computer, operating system, or computer network that is different from the MFG/PRO installation that Q/LinQ is running in. The external system could be another MFG/PRO installation.

Forwarding. the action taken when an API exports documents from one MFG/PRO module and places them in the import queue for processing by another MFG/PRO module within the same system.

Gateway. An MFG/PRO load or update program that either extracts data for sending to an external application or processes messages received from an external application. Gateways are responsible for most or all of the direct MFG/PRO database access required to implement APIs. They are internal to MFG/PRO, not exposed as part of an API. Unlike programs invoked using the Q/LinQ

user-interface emulation mode, gateway programs have no user interface and are generally better suited for reliable and faster processing of complex, variable-input data.

Guaranteed Delivery. Capability supported by various message-oriented middleware and EDI packages that assures a message sent from a source node and queue is delivered to its destination node and queue. It guarantees only delivery, not that the delivered message is received or processed by the destination application.

Guaranteed Receipt. Capability supported by various message-oriented middleware and EDI packages that assures that a message sent from a source application is received by the destination. It guarantees only receipt, not that the received message is successfully processed at the destination. Synonymous with the functional acknowledgment often implemented in EDI products.

Handle. A temporary identifier assigned to an object during a communication sessions.

Interface Message. Also known as interface document. A single package of data transported between separate applications that has integrity and is complete for both applications. The applications exchanging interface messages are often but not necessarily distributed across multiple platforms and/or network locations.

Java Messaging Service (JMS). An API from Sun Microsystems that supports messaging between computers in a network. JMS provides a common interface to standard messaging protocols and also to special messaging services in support of Java programs.

Mapped Document. A document whose format has been transformed to a format the destination application can read. See also *Document*.

Mapper. A program or a specification file used by a commercial mapping program to transform application data from native MFG/PRO format into an external format or vice versa.

Message-Oriented Middleware (MOM). A class of middleware that provides data- or message-transport services between applications when the source and destination applications are typically distributed on different platforms and development or runtime environments. Some MOM packages also include data transformation or mapping services.

Middleware. General term for off-the-shelf connectivity software that supports distributed processing at run time and application development environments. Middleware operates between the application logic and the underlying physical network. It typically handles general functions that are required to implement connectors with external applications, such as reliable message transport, message queuing, database access, distributed transaction processing, and data transformation.

MOM. See *Message-Oriented Middleware (MOM)*.

OAG. See *Open Applications Group (OAG)*.

OAGIS. See *Open Applications Group Integration Specification (OAGIS)*.

Object Management Group (OMG). A nonprofit organization dedicated to creating an interoperability standard for independently developed applications across heterogeneous networks of computers. The most significant specification released by OMG is CORBA. See *Object Request Broker (ORB)* and *Common Object Request Broker Architecture (CORBA)*.

Object Request Broker (ORB). A class of middleware that enables objects to request and provide services from or to each other over networks with different communications protocols. The objects can reside on different hardware platforms and applications. There are several emerging standards for ORB communication, chief among them Common Object Request Broker Architecture (CORBA) by the Object Management Group (OMG), and Distributed Component Object Model (DCOM) by Microsoft.

OLE/COM. Object linking and embedding/Component Object Model. Microsoft's framework for a compound document technology. Briefly, a compound document is something like a display desktop that can contain visual and information objects of all kinds: text, calendars, animations, sound, motion video, 3-D, continually updated news, controls, and so forth. Each desktop object is an independent program entity that can interact with a user and also communicate with other objects on the desktop. Part of Microsoft's ActiveX technologies, OLE takes advantage and is part of a larger, more general concept, the Component Object Model (COM) and its distributed version, DCOM). An OLE object is necessarily also a component (or COM object).

OMG. See *Object Management Group (OMG)*.

Open Applications Group (OAG). A nonprofit organization composed primarily of ERP software companies established to further multi-vendor integration by establishing and publishing specifications for the exchange of business data across applications.

Open Applications Group Integration Specification (OAGIS). The primary body of standards established and published by the OAG.

ORB. See *Object Request Broker (ORB)*.

Process Acknowledgment. Also known as process confirmation. Ability of one application to notify another that a previously sent message has been successfully processed. Process acknowledgment is stronger than the functional acknowledgment employed by some EDI standards in that it implies the message has not only been received but also processed by the target application. It is implemented in OAGIS through the BOD CONFIRM_BOD.

Protocol. The rules, formats, and functions components that a network or communication system use.

Publish-Subscribe. An asynchronous exchange of messages between applications or objects in which various parties request to receive particular types of documents or event notifications (subscribe) as other parties make them available (publish). Neither the publishing nor subscribing parties wait for a reply. Publish-subscribe usually requires middleware such as an ORB or MOM to queue the published messages and to broker communication between publishers and subscribers. Often contrasted with *Request-Reply*.

QDoc. An inbound data document to MFG/PRO in XML format that conforms to generated schemas and events files from the QGen utility.

QDoc Schema. The QDoc schema defines the structure and data types of the XML. The schema is the building block for any API request.

QGen. A tool that captures field and navigation information for an MFG/PRO or other Progress data-entry program, and generates QDoc schemas and events from the program data.

QOS. See *Quality of Service (QOS)*.

Quality of Service (QOS). A network methodology that attempts to measure, improve, and—to some extent—guarantee in advance transmission rates, error rates, and other characteristics. QOS is of special importance for continuous data transmission.

Raw Document. A document that has not been altered between the source application and destination application. See also *Document*.

Registry. A set of tables in Q/LinQ containing relatively static information about the external applications that Q/LinQ can connect with, the maps and adapters used to connect with them, and the document or message types supported for each application. Maintain the registry entries using Register External Application, Export Specification Maint, and Import Specification Maint.

Request-Reply. A synchronous exchange of messages between applications or objects in which the initiator or client makes a service request and blocks further execution until it receives a reply from the server. Request-reply is often implemented using remote procedure call (RPC) protocols or through ORBs. Often contrasted with *Publish-Subscribe*.

Session. In Q/LinQ, a single procedure that is publishing, sending, receiving, processing, archiving, or deleting documents. A Q/LinQ session can import or export one document or a group of documents in a single file or from an executing process.

Simple Object Access Protocol (SOAP). A protocol for exchanging information in a decentralized, distributed environment in XML format.

SOAP. See *Simple Object Access Protocol (SOAP)*.

Socket. A convention for connecting with and exchanging data between two program processes within the same computer or across a network. A socket represents the end point in a network connection. Sockets are created and used with a set of programming requests or function calls sometimes referred to as the socket application program interface (API). The most common sockets API is the Berkeley UNIX C language interface.

Store and Forward. An approach for passing data between applications in which each message is queued for sending by the source application and then transported asynchronously to its destination. The source application does not wait for the message to be delivered to or processed by the destination.

Synchronous Processing. Processing that is completed as part of the same application event and unit of work that initiated it. Typically, a reply or result is required from the synchronous process before execution can continue in the

initiating application. Because synchronous processes constitute the critical path of a transaction, they tend to affect application performance more than asynchronous ones.

Universal Unique Identifier (UUID). A hexadecimal number including a time stamp and a host identifier. Applications use UUIDs to identify many kinds of entities.

Winsock. A programming interface and the supporting program that handles input/output requests for Internet applications in a Windows operating system. Winsock is an adaptation for Windows of the Berkeley UNIX sockets interface

XML. See *Extensible Markup Language (XML)*.

XSL. See *Extensible Stylesheet Language (XSL)*.

XSLT. See *Extensible Stylesheet Language Transformation (XSLT)*.

Index

Numerics

36.8.1 106, 131, 132, 149, 162, 166
36.8.1.1 106, 131, 132, 149, 162, 166
36.8.1.2 106, 131, 132, 149
36.8.1.3 106, 131, 132, 162, 166
36.8.1.20 159, 171
36.8.2 106, 131, 132, 149
36.8.3 106, 131, 132, 162, 166
36.8.10 162
36.8.22.1 153

A

acceptMsg 75
acknowledgment
 Q/LinQ communications 15
 qqAckReceivedDoc 64
adapters
 export 14
 import 14
 messaging API 60
 MQSeries 16
 publishing API 140
 stream API controller 17
 stream API services 42
application program interface (API)
 mapping 106
 message-oriented middleware (MOM) 16
 messaging 59
 processing 161
 publishing 140
 stream controller 16
 stream services 42

C

Code Mapping Inquiry 176
Code Mapping Maintenance 176
communication APIs
 See also stream API, messaging API

message-oriented middleware (MOM) 16
messaging 59
overview 12
 stream controller 16
 stream services 42
CONFIRM_BOD control-tag file 129
connect 73
CONNECTDATA, ConnectData parameters 95
constants, predefined 65
control tags 125
CTRLHEAD, CtrlHead parameters 98

D

Data Synchronization Setup menu 153
direct database access
 qqBuildCIMLine 122
 qqGetRawDocCtrl 114
 qqGetRawDocElement 117
 qqGetRawDocLine 116
 qqLogMapException 123
 qqMappedCode 113
 qqPutMappedDocCtrl 108, 118
 qqPutMappedDocElement 121
 qqPutMappedDocLine 120
disconnect 78
DOM Builder
 XML publishing 143, 156
dump/load text file 124

E

errors
 mapping API 132
 mapping API text file dump/load 137
 processing API 169, 170
Export Specification Maint
 input/output card parameters 131
 mapping API 106

F

file formats

- control tag for mapping API 125
- error data for mapping API 132
- mapped data for mapping API 130
- raw data for mapping API 129

forwardMsg 72

G

gateway programs 162

- parameters 164
- qqGetNextDocElement 166
- qqGetNextDocLine 168
- qqLogOutputMsg 170
- qqLogProcException 169
- qqMappedCode 171

I

identify source application 175

Import Specification Maint

- gateway programs 162
- input/output card parameters 131
- mapping API 106

.ini file format, publishing API 150

IP address, Q/LinQ use of 14

iQqSender 72

isAlive 79

J

Java classes, messaging API 69, 70

L

Logger, public class 70

M

map code values 176

mapping API

See also direct database access

See also text file dump/load

overview 106

program decomposition 107, 133

menu reference conventions 4

message-oriented middleware (MOM) API 16

messaging API 14, 59

acceptMsg 75

adapter 60

connect 73

CONNECTDATA, ConnectData 95

CTRLHEAD, CtrlHead 98

disconnect 78

functional description 63

iQqSender 72

isAlive 79

Java classes 69, 70

limitations 15

overview 14

program decomposition 59, 65

qqAckConnect 63, 98

qqAckReceivedDoc 64, 91

QqApi 69, 70

qqCheckConnect 94

QqConnectionActive 70

QqConnectionPassive 70

qqDisconnect 63, 92

qqReceiveConnect 63, 98

qqReceiveDocCtrl 63, 87

qqReceiveDocData 64, 89

qqSendConnect 63, 74

qqSendDocCtrl 63, 85

qqSendDocData 64, 86

receiveMsg 77

return codes 66

routine libraries 60

security issues 16

testing communication 67

mgckapp.p 174

mgqqapp.i 175

MOM. *See* message-oriented middleware (MOM) API

MQSeries 16

O

OAGIS BODs

CONFIRM_BOD 129

mapping 125

publishing to 151

P

Process Import Documents 162

processing API 161

adapter example 163

gateway programs 162

gateway programs parameters 164

program decomposition 163

qqGetNextDocElement 166

qqGetNextDocLine 168

qqLogOutputMsg 170

qqLogProcException 169

qqMappedCode 171

process-to-process communications 14

program decomposition

mapping API adapter 133

mapping procedure 107

messaging API adapter 60

- processing API 163
- publishing API 141
- stream API controller 17
- stream API services 42
- Progress
 - mapping procedures 107
 - stream API controller, use of 16
 - stream API services, use of 42
 - use with publishing adapter 140
- publishing API 140
 - adapter example 141, 142
 - OAGIS BODs 151
 - program decomposition 141
 - qqBeginInterfaceDoc 145
 - qqExtractRecordData 153
 - qqMappedCode 159
 - qqpubfwd.p 141
 - qqPublishInterfaceDoc 155
 - qqPublishXMLDoc 156
 - triplet format 140
 - XML format 142, 156
- Q**
- Q/LinQ Control
 - System ID value 145, 157
 - time zone 128
- qqAckConnect 63, 98
- qqAckReceivedDoc 64, 91
- QqApi
 - constructor 69
 - Java class 69
 - parameters 70
 - public class Logger 70
- qqapiunix.h 68
- qqapiwin.h 68
- qqBeginInterfaceDoc 145
- qqBuildCIMLine 122
- qqCheckConnect 94
- qqCloseSrcDest 41
- qqconn.c 68
- QqConnectionActive
 - Java class 70
 - parameters 71
- QqConnectionPassive
 - Java class 70
 - parameters 71
- qqDisconnect 63, 92
- qqDumpRawDocFiles 135
- qqExtractRecordData 153
- qqgetarg.c 68
- qqGetDocCtrl 52
- qqGetDocData 56
- qqGetNextDocElement 166
- qqGetNextDocLine 168
- qqGetRawDocCtrl 114
- qqGetRawDocElement 117
- qqGetRawDocLine 116
- qqLoadMapExceptionFile 137
- qqLoadMappedDocFile 136
- qqLogMapException 123
- qqLogOutputMsg 170
- qqLogProcException 169
- qqMappedCode 113, 159, 171, 176
- QqMomAdapter 16
- qqOpenSrcDest 20
- qqprctrl.c 68
- qqpubfwd.p 141
- qqPublishInterfaceDoc 155
- qqPublishXMLDoc 156
- qqPutDocCtrl 46
- qqPutDocData 50
- qqPutMappedDocCtrl 108, 118
- qqPutMappedDocElement 121
- qqPutMappedDocLine 120
- qqrdf.c 68
- qqReadDocCtrl 33
- qqReadDocData 40
- qqReceiveConnect 63, 82, 98
- qqReceiveDocCtrl 63, 87
- qqReceiveDocData 64, 89
- qqrecv 68
- qqrecv, test C program 60, 68
- QqRecv, test Java program 60, 68
- qqrecv.exe 68
- QqRecvSimple, test Java program 60, 69
- qqsend 68
- qqsend, test C program 60, 68
- QqSend, test Java program 60, 68
- qqsend.exe 68
- qqSendConnect 63, 74
- qqSendDocCtrl 63, 85
- qqSendDocData 64, 86
- QqSendSimple, test Java program 60, 69
- qqStartSession 44
- qqStopSession 58
- qqstrapi.p 42
- qqvarray.c 68
- qqWriteDocCtrl 24
- qqWriteDocData 32
- QXtend Inbound
 - overview 140
- R**
- receiveMsg 77

Register External Application
 check registered applications 174
 DocRev value 26, 35
 DocStd value 25, 35
 DocType value 26, 35
 gateway programs 162
 input/output card parameters 131
 mapping API 106
 return codes 66
 routine libraries, messaging API 60

S

security of messaging API 16
 SOAP 140
 sock_stream 14
 static variables, public 65
 stream API 13
 controller 13, 16
 overview 13
 program decomposition, controller 17
 program decomposition, services 42
 qqCloseSrcDest 41
 qqGetDocCtrl 52
 qqGetDocData 56
 qqOpenSrcDest 20
 qqPutDocCtrl 46
 qqPutDocData 50
 qqReadDocCtrl 33
 qqReadDocData 40
 qqStartSession 44
 qqStopSession 58
 qqstrapi.p 42
 qqWriteDocCtrl 24
 qqWriteDocData 32
 services 13, 42
 Synchronization Profile Maintenance 153

T

TCP sockets, Q/LinQ use of 14
 test programs
 QqRecvSimple.java 69
 QqRecvSimple.java 60
 qqsend.c, qqrecv.c 60, 68
 QqSend.java, QqRecv.java 60
 QqSend.java, QqRecv.java 68
 QqSendSimple.java 60, 69
 wQqRecvSimple.java 69
 wQqSend.java, wQqRecv.java 60
 wQqSendSimple.java 60
 text file dump/load 124
 CONFIRM_BOD BOD 129
 control tags 125
 error file format 132
 mapped-data file format 130
 qqDumpRawDocFiles 135
 qqLoadMapExceptionFile 137
 qqLoadMappedDocFile 136
 raw-data file format 129
 triplet format 140

U

utilities
 check application registry 174
 identify source application 175
 map code values 176

W

wQqRecv, Java test program 60
 wQqRecvSimple, Java test program 60
 wQqSend, Java test program 60
 wQqSendSimple, Java test program 60

X

XML format
 publishing 140