



Technical Reference

DEVELOPER Q/LINQ



78-0408A
MFG/PRO Version 9.0
Printed in the U.S.A.
March 1999

This document contains proprietary information that is protected by copyright. No part of this document may be photocopied, reproduced, or translated 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.

Some states do not allow the exclusion of implied warranties or the limitation or exclusion of liability for incidental or consequential damages, so the above limitations and exclusion may not be applicable.

PROGRESS® is a registered trademark of Progress Software Corporation. Windows™ is a trademark of Microsoft Corporation.

UNIX® is a registered trademark of UNIX System Laboratories, Inc.

Windows™ is a trademark of Microsoft Corporation.

Windows NT® and MS-DOS® are registered trademarks of Microsoft Corporation.

Mercator is a trademark of TSI International Software Ltd.

MQSeries® is a registered trademark of IBM Corporation.

MFG/PRO® is a registered trademark of QAD Inc.

Copyright © 1999 by QAD Inc.

78-0408A

QAD Inc.

6450 Via Real

Carpinteria, California 93013

Phone (805) 684-6614

Fax (805) 684-1890

Contents

ABOUT THIS GUIDE	1
What Is in This Guide?	2
Other 9.0 Documentation	2
Online Help	2
QAD Web Site	3
Conventions	3
Windows Keyboard Commands	3
Character Keyboard Commands	4
 CHAPTER 1 API REFERENCE OVERVIEW	 7
Introduction	8
File Locations	8
Programming Conventions	9
 CHAPTER 2 COMMUNICATIONS API REFERENCE	 11
Overview	12
Stream API	13
Messaging API	13
MOM API	15
Stream API for PROGRESS	16
Program Decomposition of Stream API Adapter	16
qqOpenSrcDest	19
qqWriteDocCtrl	22
qqWriteDocData	29

qqReadDocCtrl	30
qqReadDocData	35
qqCloseSrcDest	36
Messaging API	37
Program Decomposition of Messaging API Adapters	38
Messaging API Functional Description	41
Predefined Constants and Public Static Variables	42
Return Codes	43
Using the Test Programs	44
Messaging API: Java Classes and Methods	46
C and Java Routine Descriptions	47

CHAPTER 3 MAPPING API REFERENCE 71

Overview	72
Direct Database Access	72
Program Decomposition of Mapping API Adapter	73
Mercator Adapter	75
Direct Database Mapping API	77
Text File Dump/Load	84
Text File Formats	84
Text File Dump/Load Mapping API	93

CHAPTER 4 PUBLISHING API REFERENCE 99

Overview	100
Program Decomposition of Publishing API Adapter	101
qqBeginInterfaceDoc	102
qqBuildDocElement	106
qqPublishInterfaceDoc	110

CHAPTER 5 PROCESSING API REFERENCE..... 111

Overview	112
Gateway Programs	112
Program Decomposition of Processing API Adapter	113
Gateway Parameters	114

Gateway Procedures	115
CHAPTER 6 Q/LINQ UTILITIES.....	121
Check Registry	122
Identify Source Application	123
Q/LINQ GLOSSARY	125
INDEX	131

About This Guide

What Is in This Guide? 2

Windows Keyboard Commands 3

Character Keyboard Commands 4

Routing Maintenance (Main Screen)

Routing Code:	10-15000	NONAD (75) COOLING
Operation:	20	
Standard Operation		
Work Center	1030	INSPECTION, ALL SITES
Machine		
Description	INSPEC PER PROC-005	
Machines per Op	1	Milestone P
Overlap Units	1	
Queue Time	1.0	
Wait Time	0.0	
Setup Time	0.0	

Route by Production Route

What Is in This Guide?

This guide provides detailed technical information for Developer Q/LinQ.

This information can be used by developers to build custom APIs that enable Q/LinQ to communicate with any external system, 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 systems.

Other 9.0 Documentation

- For an overview of new features and software updates, see the *9.0 Release Bulletin*.
- For software installation instructions, refer to the *9.0 Installation Guides*.
- For instructions on installing Q/LinQ, refer to the *Q/LinQ Installation Guide*.
- For instructions on navigating the MFG/PRO environment, refer to the *9.0 User Interface Guide*.
- For information on using MFG/PRO, refer to the *9.0 User Guides*.
- For technical details, refer to the *9.0 File Relationships* and *9.0 Database Definitions*.
- To review MFG/PRO program screens, refer to the *Screen Book, Volumes 1–3*.
- To view documents online in PDF format, see the *9.0 Documents on CD*.

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, browses, and control files do not have procedure help.

For information on using the help system, refer to the *User Interface Guide*.

QAD Web Site

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

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

To obtain a QAD Web account, go to:

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

The QAD Web site also has information about training classes and other services that can help you learn about MFG/PRO.

Conventions

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:

- MFG/PRO screen captures reflect the Windows interface.
- References to keyboard commands are generic. For example, choose Go refers to F2 in the Windows interface and to F1 in the character interface. The following tables identify the keyboard commands for the Windows and character interfaces.

Windows Keyboard Commands

Navigation Commands	Keyboard Entry	Description
Go	F2	Moves to next frame.
End	Esc	Exits a frame, program, or menu.
Previous	F9 or Up Arrow	Retrieves previous record in a key data field.
Next	F10 or Down Arrow	Retrieves next record in a key data field.
Enter	Enter	Moves to next field within a frame.
Tab	Tab	Moves to next field within a frame.
Back Tab	Shift+Tab	Moves back one field within a frame.
Exit	Alt+X	Closes a program.

Navigation Commands	Keyboard Entry	Description
Run	Ctrl+R	Starts a program by name.
Save (object)	F12	In a key frame, moves to data entry; in data entry, saves and returns to key frame.
Print (object)	Ctrl+P	Prints browse or maintenance information.

Help Commands	Keyboard Entry	Description
Field Help	F1	Opens help on current field.
Procedure Help	Shift+F1	Opens help on current program.
Browse	Alt+F1	Displays choice of records.
Look-Up Browse	Alt+F2	Displays choice of records.
About	Ctrl+F1	Displays the program name.
Browse Options	F7	Opens the browse options window.
Browse Options Toggle	Alt+F	Turns the browse options on and off.
Browse Graph	Shift+F11	Opens the browse graphing window.
Field Name	Ctrl+F	Displays the field name.

Edit Commands	Keyboard Entry	Description
Delete Record	F5	Deletes an open record.
Cut	Ctrl+X	Cuts a field or selection to clipboard.
Copy	Ctrl+C	Copies a field or selection to clipboard.
Paste	Ctrl+V	Pastes data from the clipboard.

Character Keyboard Commands

Navigation Commands	Keyboard Entry	Control Key Entry	Description
Go	F1	Ctrl+X	Moves to next frame.
End	F4	Ctrl+E	Exits a frame, program, or menu.

Navigation Commands	Keyboard Entry	Control Key Entry	Description
User Menu	F6	Ctrl+P	Displays list of user-selected programs.
Previous	F9 or Up Arrow	Ctrl+K	Retrieves previous record in a key data field and scrolls up in look-up browses.
Next	F10 or Down Arrow	Ctrl+J	Retrieves next record in a key data field and scrolls down in look-up browses.
Enter	Enter		Moves to next field within a frame.
Tab	Tab		Moves to next field within a frame.
Back Tab	Shift+Tab	Ctrl+U	Moves back one field within a frame.
Menu Bar (object)	Esc, M		Accesses the menu bar.
Save (object)	F12		In a key frame, moves to data entry; in data entry, saves and returns to key frame.
Print (object)	Ctrl+P		Prints browse or maintenance information.

Help Commands	Keyboard Entry	Control Key Entry	Description
Field Help	F2	Ctrl+W	Opens help on current field.
Procedure Help	F2	Ctrl+W	Opens help on current program.
Look-Up Browse	F2	Ctrl+W	Displays choice of records.
Browse Options	F7		Opens the browse options window.
Browse Options Toggle	Alt+F		Turns the browse options on and off.
Field Name	Ctrl+F	Ctrl+F	Displays the field name.

Edit Commands	Keyboard Entry	Control Key Entry	Description
Insert	F3	Ctrl+T	Enables text insertion.
Delete Record	F5	Ctrl+D	Deletes an open record.

Edit Commands	Keyboard Entry	Control Key Entry	Description
Recall (standard)	F7	Ctrl+R	Recalls last saved value in a field.
Cut	F8		Clears a field.
Copy	F11	Ctrl+B	Copies a field.
Paste	F11	Ctrl+B	Inserts value that you copied.
Multiple Copy (standard)	F12	Ctrl+A	Copies values from one or more fields and pastes them into the same fields of another record.
Clear Date	Shift+?		Clears the value in date fields.

CHAPTER 1

API Reference 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**

Routing Code: 10-15000 NONAD(75) CDB IN

Operation: 20

Standard Operation: 1030 INSPECTION, ALL SITE

Work Center: Machine

Description: INSPIC PER PROX-005

Machines per Op: 1 Milestone: 1

Overlap Units: 1

Queue Time: 1.0

Wait Time: 0.0

Setup Time: 0.0

Route by Production Route

Seq	Op	Op Desc	Stn	Time	Time	Time	Time	Time	Time
1	10	1030	1030	1.0	0.0	0.0	0.0	0.0	0.0
2	20	2030	2030	1.0	0.0	0.0	0.0	0.0	0.0
3	30	3030	3030	1.0	0.0	0.0	0.0	0.0	0.0
4	40	4030	4030	1.0	0.0	0.0	0.0	0.0	0.0
5	50	5030	5030	1.0	0.0	0.0	0.0	0.0	0.0
6	60	6030	6030	1.0	0.0	0.0	0.0	0.0	0.0
7	70	7030	7030	1.0	0.0	0.0	0.0	0.0	0.0
8	80	8030	8030	1.0	0.0	0.0	0.0	0.0	0.0
9	90	9030	9030	1.0	0.0	0.0	0.0	0.0	0.0
10	100	10030	10030	1.0	0.0	0.0	0.0	0.0	0.0

Introduction

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

- Build custom APIs that enable Q/LinQ to communicate with any external system
- 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 systems

Use the information in this book in conjunction with *MFG/PRO User Guide Volume 10B: External Interfaces*. That volume 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, as well as how to use the Universal Logistics API to communicate with an external logistics system.

File Locations

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

Table 1.1
File Locations

Directory	Description
doc	Language-specific subdirectories containing the <i>Developer Q/LinQ Technical Reference</i> in portable document format (PDF).
lib	Sub-directories 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	Sub-directories 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.

CHAPTER 2

Communications API Reference

This chapter discusses the APIs that support communication between Q/LinQ and external systems.

Overview 12

Stream API for PROGRESS 16

Messaging API 37

The screenshot displays a window titled "14.13.2 Routing Maintenance (Main Screen)". It contains a table with the following data:

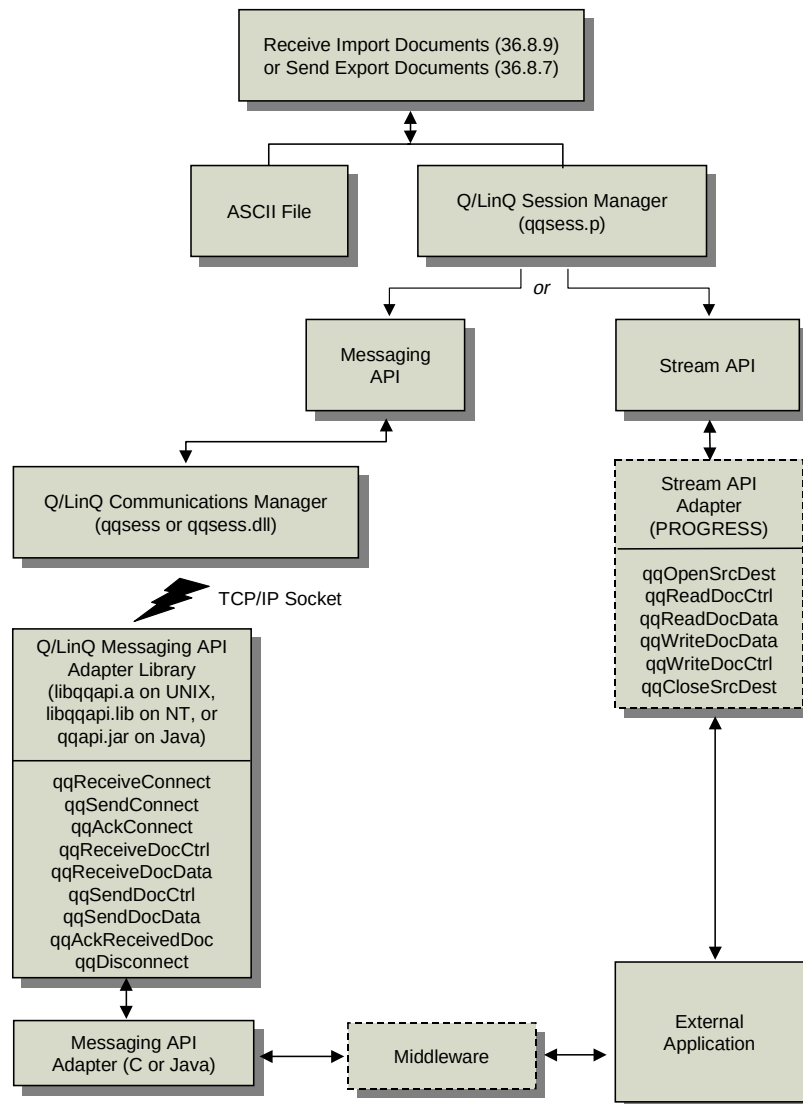
Routing Code	Operation	Standard Operation	Work Center	Description	Machines per Op	Overlap Units	Queue Time	Wait Time	Setup Time
10-15000	20	10.00	INSPEC	INSPEC PER PROQ-000	1	1	1.0	0.0	0.0

Below the table, it indicates "Rout by Production Build" and "Rout by Time".

Overview

Three Q/LinQ APIs support communication with an external system's processes: the stream, messaging, and message-oriented middleware (MOM) APIs. Figure 2.1 represents an information flow for these three APIs.

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



Stream API

When Q/LinQ always initiates the connections and read and write operations, use the stream API for interfaces that connect directly to other PROGRESS applications or databases.

The stream API includes exit points to PROGRESS 4GL adapter programs that:

- Connect to external systems
- Disconnect from external systems
- Read interface data
- Write interface data

Messaging API

Q/LinQ provides a messaging API to handle socket-based data-exchange protocols as defined by Q/LinQ. This messaging API enables direct connectivity with non-PROGRESS applications via sockets across a TCP/IP network.

This process-to-process communications interface requires that programmers design and write import or export adapters.

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

Programmers have limited freedom in designing the import adapters that communicate with the external system. Options include reading text files or calling procedures inside an external adapter. These custom adapters connect to Q/LinQ via the messaging API.

Use the messaging API when:

- The external system to connect is not a PROGRESS program; for example, IBM's MQSeries message-oriented middleware.
- The programmer lacks a PROGRESS development license or PROGRESS 4GL skills.
- The external system, rather than Q/LinQ, initiates network connections.

- Q/LinQ and the external system reside on different host computers inside the same enterprise and middleware does not mediate communications between them.

Q/LinQ communicates with external systems using TCP sockets of type `SOCK_STREAM`. In the UNIX environment, Berkeley Software Design (BSD) sockets are used. Windows sockets 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, which lets you maintain and compile them separately.

▶ See *User Guide Volume 10B: External Interfaces*.

When an application using interprocess communication is registered with Q/LinQ, you must specify a static IP address and port number. Q/LinQ uses this information to connect to a socket and communicate through adapters with the external system. Based on how the external system is registered with Q/LinQ, Q/LinQ can actively initiate the connection or passively wait for the external system to connect. However, Q/LinQ does not start or stop external-system 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 system received each packet of data in the document.

Note This communications acknowledgment has no relation to Q/LinQ's autoacknowledgment feature. The first is a low-level call within the messaging API after transmission of each packet of data. The second is sent when the entire document has been received.

Other than information 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 system 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.

◆ See *User Guide Volume 10B: External Interfaces*.

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.

Warning Never use the messaging API to send data outside the company firewall or across public networks if security or reliability is a concern.

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

- Provides no guaranteed delivery or other quality of service (QOS) options
- Provides no access-security checking or data encryption
- Provides no warm recovery, restart, or resend capability after network failures
- Requires predefined and static rather than dynamic IP addresses and ports to establish connections
- Cannot multi-thread multiple interface connections through the same socket and must have a separate dedicated port for each concurrent connection

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 Q/LinQ's messaging API.

MOM API

Q/LinQ provides an adapter to IBM's MQSeries, the MOM Q/LinQ supports. The adapter, `QqMomAdapter`, is written in Java. 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.

◆ See *User Guide Volume 10B: External Interfaces* for directions on using the MOM adapter.

Stream API for PROGRESS

Use the stream API for direct modes of transport, such as writing to or reading from files that do not conform to the Q/LinQ format or transferring information via an E-mail system or file transfer protocol (FTP).

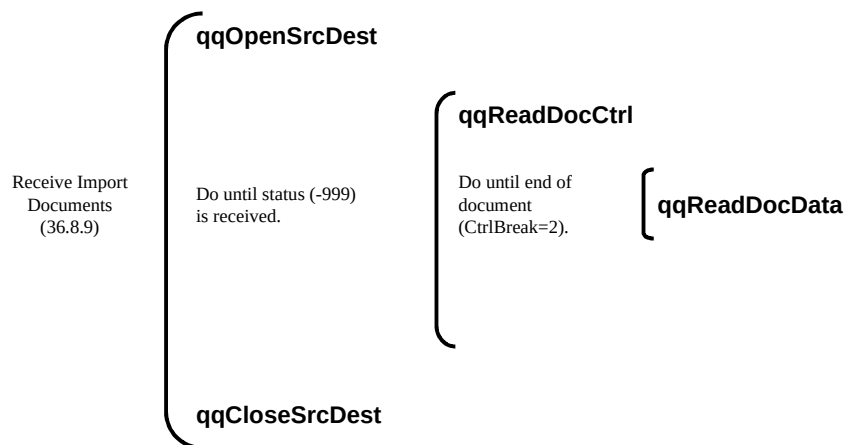
The stream API is a set of calls from Q/LinQ to a custom PROGRESS 4GL procedure that uses the subroutines listed in Figure 2.2 and Figure 2.3.

Program Decomposition of Stream API Adapter

When importing documents, the stream API:

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

Fig. 2.2
Stream API Import
Call Logic



When exporting documents, the stream API:

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

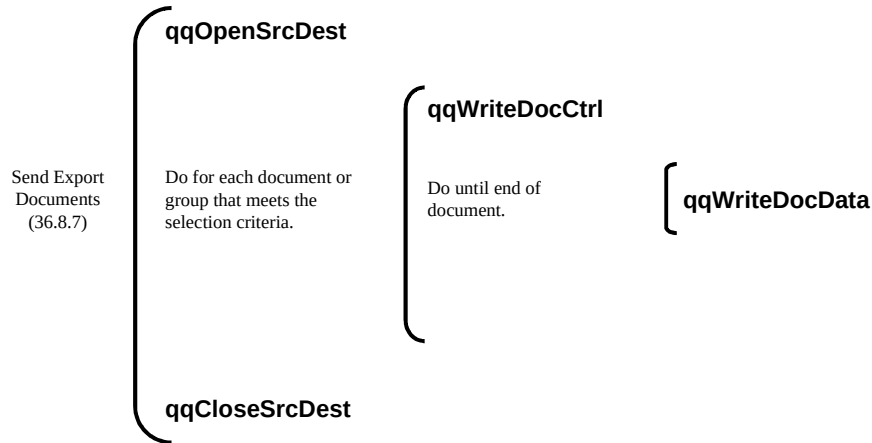


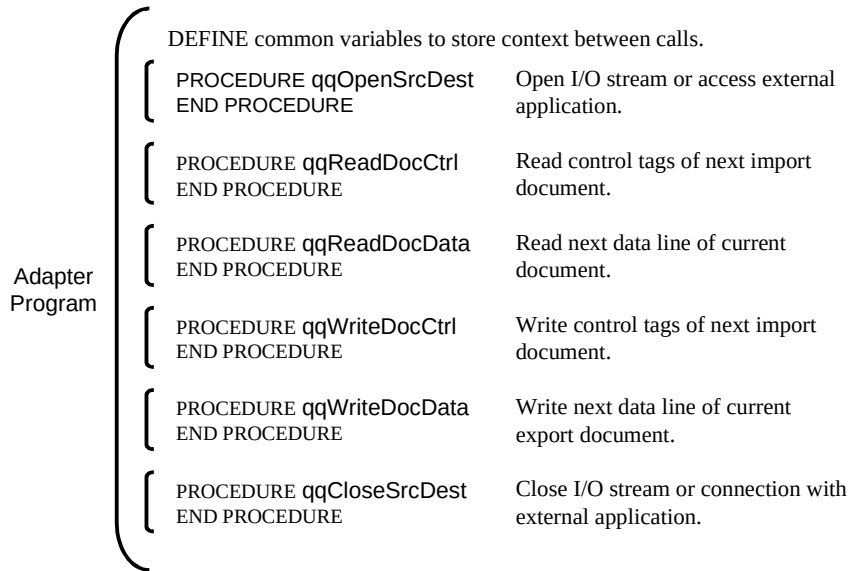
Fig. 2.3
Stream API Export
Call Logic

Figure 2.4 shows the general structure of a sample stream API adapter that the Q/LinQ stream API 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 uses the internal procedures in Figure 2.2 and Figure 2.3.

Fig. 2.4
Adapter-Initiated
Export Connection



qqOpenSrcDest

Parameter Name	Type	Description	I/O	Req
AppID	x(12)	External application to connect to. 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 required by the API to open a session with the external application, such as an authentication ID or name. Possible uses are: • A password or security-related key, although it is not encrypted. • For adapters that directly access a non-MFG/PRO database rather than directly call an external application, 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: • Export (0) • Import (1) Each connection must be of either one type or the other, never both.	I	Y
CurrSessID	x(8)	The Q/LinQ interface session that is communicating with the adapter.	I	Y

qqOpenSrcDest — (1 of 3)

Parameter Name	Type	Description	I/O	Req
MsgAckLvl	x(8)	Not implemented. The QOS 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. Optional and relevant only to some middleware products. The QOS parameter describing the level of synchronization of an exported document with respect to the destination application. 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: • 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)	Not implemented. Relevant only to some middleware products. The QOS parameter describing the level of security. The security level can be: • 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 QOS parameter that overrides middleware settings by application or document.	I	N

qqOpenSrcDest — (2 of 3)

Parameter Name	Type	Description	I/O	Req
MsgRouting	x(8)	Not implemented. Network routing of a document from source to a destination (such as Queue ID). MOM QOS 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 or put documents from or to.	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
MsgPersist	x(8)	Not implemented. Relevant only to some middleware products. The QOS 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
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.	O	N

qqOpenSrcDest — (3 of 3)

qqWriteDocCtrl

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
UoWorkType	x(1)	Defines how the documents in the group (see GroupID) are processed in relation to each other. • 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. • Series (1). The documents in a series 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. • Batch (2). The documents in a batch group must be processed in the order that they were imported or exported. If one contains an error, none of the documents are committed. • Partial group (9). The document is a member of a series or batch group that was previously partially imported or exported and follows the first document. 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.	I	Y
GroupID	integer	Identifier of a set of documents that have a certain processing dependency (see UoWorkType).	I	Y

qqWriteDocCtrl — (1 of 7)

Parameter Name	Type	Description	I/O	Req
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
DocType	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.	I	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.	I	Y

qqWriteDocCtrl — (2 of 7)

Parameter Name	Type	Description	I/O	Req
TradPtrID	x(32)	Identifies the trading partner receiving or sending the document. Also use this parameter to differentiate between source applications that communicate using the same middleware in MOM environments. 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 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. When Q/LinQ's 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.	I	N
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. - For import documents, the API that publishes the documents from the external application to MFG/PRO supplies the value.	I	N
MfgproSite	x(8)	Identifies the MFG/PRO site associated with the export document came from. Established when the document was published.	I	N

qqWriteDocCtrl — (3 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. This value originates in MFG/PRO. - For export documents, the API that publishes the documents from MFG/PRO to Q/LinQ supplies the MFG/PRO key field value. - For import documents, the API that publishes the documents from the external system to MFG/PRO supplies the 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 send to or receive from the external system. - 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.	I	N
OrigDocId	x(8)	Identifies an original source document previously exported from Q/LinQ and passed as a reference from the sending application (such as an acknowledgment).	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

Parameter Name	Type	Description	I/O	Req
NumForm	x(8)	Describes the format used to represent decimal numbers in the interface document, using the values supported by PROGRESS. Possible values are: - American, where the comma separates thousands and the period is a decimal point - European, where 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
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

qqWriteDocCtrl — (5 of 7)

Parameter Name	Type	Description	I/O	Req
MsgAckLvl	x(8)	Not implemented. The QOS 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 QOS 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: • 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)	Not implemented. Relevant only to some middleware products. The QOS parameter describing the level of security. The security level can be: • 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 QOS parameter that overrides middleware settings by application or document.	I	N

qqWriteDocCtrl — (6 of 7)

Parameter Name	Type	Description	I/O	Req
MsgRouting	x(8)	Not implemented. Network routing of a document from source to a destination (such as Queue ID). MOM QOS 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 or put documents from or to.	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
MsgPersist	x(8)	Not implemented. Relevant only to some middleware products. The QOS 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.	O	N

qqWriteDocCtrl — (7 of 7)

qqWriteDocData

Parameter Name	Type	Description	I/O	Req
DestAppID	x(12)	Application to which a connection is requested.	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
CtrlBreak	x(1)	• 2: The data represents the end of the document. • 1: The data represents the end of the current line in the document. • 0: The data continues to the next line.	I	Y
DocLength	integer	Number of characters included in DocData.	I	Y
DocData	x(900)	A packet of data from the exported document.	I	Y
ResultStatus	integer	Indicates success or failure of the request.	O	N

qqReadDocCtrl

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.	O	N
UowType	x(1)	Defines how the documents in the group (see ExtGroupID) are processed in relation to each other. - 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. - 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. - 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. - Partial group (9). The document is a member of a series or batch group that follows the first document in the group. 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.	O	Y
ExtGroupID	integer	The ID of the group that the document is part of. Assigned by the external system and not used by Q/LinQ.	O	Y

qqReadDocCtrl — (1 of 5)

Parameter Name	Type	Description	I/O	Req
ExtGroupSeq	integer	The ordinal position within the group that the document belongs to. It is assigned by the external system and not used by Q/LinQ.	I	N
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
DocType	x(16)	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 — (2 of 5)

Parameter Name	Type	Description	I/O	Req
TradPtrID	x(32)	Identifies the trading partner receiving or sending the document. Also use this parameter to differentiate between source applications that communicate using the same middleware in MOM environments. 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. When Q/LinQ's 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.	O	N
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. - For import documents, the API that publishes the documents from the external application to MFG/PRO supplies the value.	O	N
MfgproSite	x(8)	Identifies the MFG/PRO site the import document is directed to.	O	N

qqReadDocCtrl — (3 of 5)

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. This value originates in MFG/PRO. - For export documents, the API that publishes the documents from MFG/PRO to Q/LinQ supplies the MFG/PRO Key field value. - For import documents, the API that publishes the documents from the external system 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 system. - 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).	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

qqReadDocCtrl — (4 of 5)

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: - American, where the comma separates thousands and the period is a decimal point - European, where 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 system.	O	N
TimeCreate	nn:nn:nn	Time of day the document was created by the external system.	O	N
TimeZone	integer	The time zone of DateCreate and TimeCreate.	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
CtrlTot2	integer	Not used on received documents.	O	N
CtrlTot3	integer	Not used on received documents.	O	N

qqReadDocCtrl — (5 of 5)

qqReadDocData

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.	O	N
CtrlBreak	x(1)	• 2: The data represents the end of the document. • 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

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.	O	N

Messaging API

Using a messaging API to import documents from a non-PROGRESS external system into Q/LinQ comprises 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 an object.
- 3 Receive documents. The external system or MOM sends the document data in packets to Q/LinQ via the adapter that is tailored to fit the external system. 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 system that it successfully received or failed to receive each packet.

Note This communications acknowledgment and the one in step 4, below, has no relation to Q/LinQ's 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 system comprises four steps very similar to the import steps.

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

♦ See *User Guide Volume 10B: External Interfaces*.

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

- Several libraries of routines
 - UNIX (`libqqapi.a`)
 - Java (`qqapi.jar`)
 - Windows (`libqqapi.lib`)
- Include files for UNIX and Windows
- Test C programs: `qqsend.c` and `qqrecv.c`
- Test Java programs: `QqSend.java`, `QqRecv.java`, `wQqSend.java`, and `wQqRecv.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 44.
- Information specific to Java programs begins on page 46.

The tables beginning on page 47 contain parameter descriptions and data types for both the C functions and data structures and the Java methods and classes.

Program Decomposition of Messaging API Adapters

Messaging adapters are not an actual 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).

Tip
Q/LinQ does not start or stop the adapters.

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 system using the socket

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 2.5 through Figure 2.8 illustrate this structure. In all four combinations, the sections of the API follow the same general pattern.

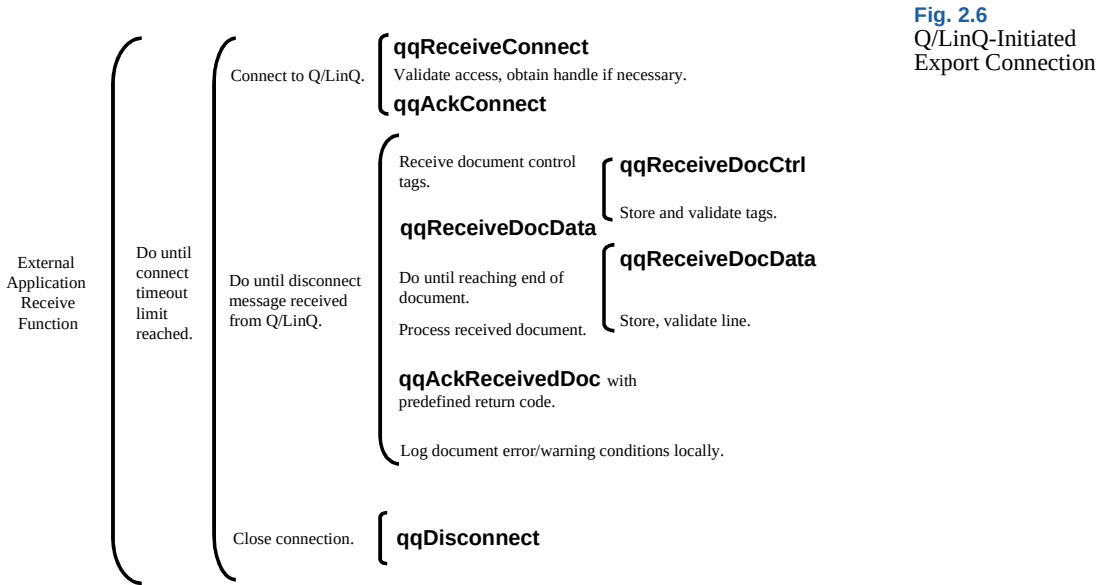
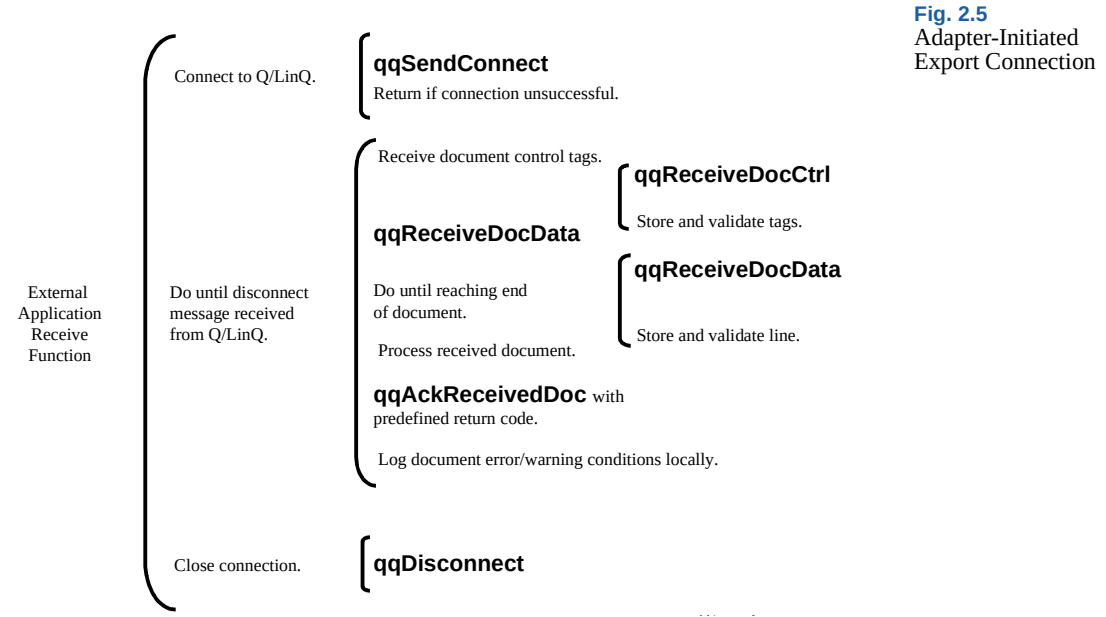


Fig. 2.7
Adapter-Initiated
Import Connection

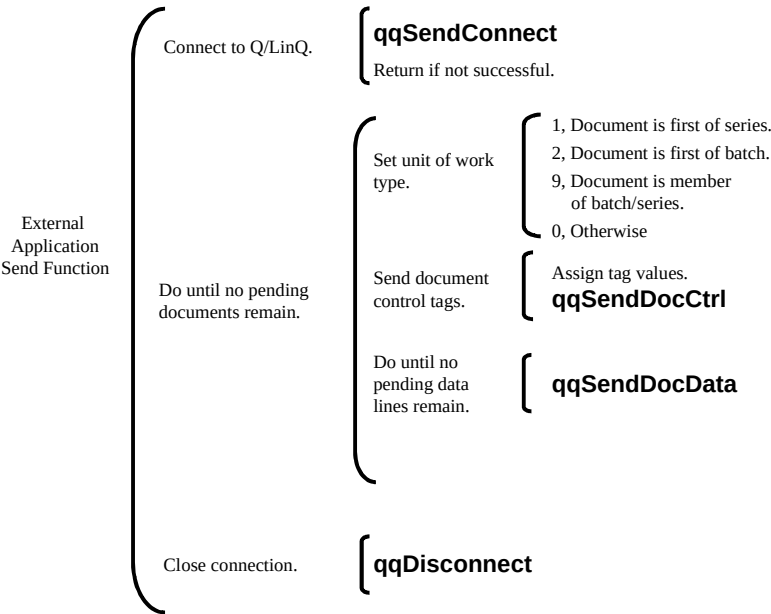
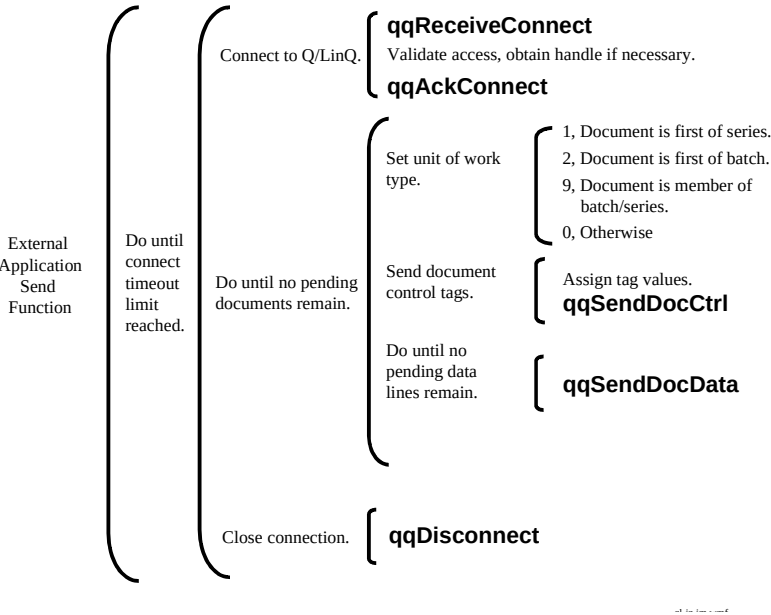


Fig. 2.8
Q/LinQ-Initiated
Import Connection



Messaging API 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 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 the document is the last in its group.

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 Data Packet Received

When receiving the document data, Q/LinQ requires an acknowledgment message after every 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.

Close Connection

When the session has completed sending or receiving, socket connections must close. The side sending documents, whether 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.

◆ See “Loop Through Documents to Send or Receive” on page 41.

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.

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.

Table 2.1
`qqapiunix` and
`qqapiwin` Con-
stants and `QqApi`
Public Static Inte-
ger Variables

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.

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.

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.

Return Code	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
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

Table 2.2
Return Codes of
qqapiunix,
qqapiwin, and
QqApi

Return Code	Description	Value
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

▶ See *User Guide Volume 10B: External Interfaces* for information on Java 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 test programs can:

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

▶ See *User Guide Volume 10B: External Interfaces* for more on the C test programs.

`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 file.
- `qqvarray.c` reads the Q/LinQ file into a memory buffer for easier processing.
- `qqrdfile.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.

Compiling Test Programs

Compile `qqsend.c` and `qqrecv.c` in a UNIX environment with an ANSI C compiler. Use the make files `qqsend.mak` and `qqrecv.mak` to compile in a UNIX environment. The link library is `libqqapi.a`.

Compile `qqsend.c` and `qqrecv.c` in a Windows environment with the Microsoft Visual C+ V 5 compiler.

If compiling in Visual C++, define preprocessor variables `OS_TYPE` as 2 and `PARM_TYPE` as 1. The link libraries `ws2_32.lib` and `libqqapi.lib` are required. Use the runtime library multi-threaded.

Running Test Programs

Run `qqsend.c` and `qqrecv.c` from either a UNIX prompt or a Windows command prompt.

Messaging API: Java Classes and Methods

Public Class QqApi

Extends object.

This class is the Java implementation of the messaging API for MFG/PRO. 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 typically use this API.

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)
```

Parameters of QqApi

Parameter	Description
log	Logger instance for log file or display messages.

Public Class Logger

Extends object.

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.

C and Java Routine Descriptions

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

- qqSendConnect 48
- CONNECTDATA Parameters 50
- qqReceiveConnect 53
- qqAckConnect 55
- qqSendDocCtrl 56
- CTRLHEAD Parameters 57
- qqReceiveDocData 65
- qqAckReceivedDoc 67
- qqDisconnect 68

qqSendConnect

This routine opens a socket on the local machine and requests a connection with Q/LinQ when the external system initiates 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 Type
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	int	int
PollWait	Maximum number of seconds to wait for an acknowledgment message from Q/LinQ after requesting the connection and/or after sending each packet of document data.	I	Y	int	int
Cdata	Q/LinQ connection parameters. See “CONNECTDATA Parameters” on page 50.	I	Y	CONNECT DATA	connect Data

Return Codes

Return Code	Description	Value
QQAPI_SUCCESS	Successful connection.	0
QQAPI_INVALID_IP	Failed to establish a connection with Q/LinQ due to an invalid IP address.	2472
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_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

CONNECTDATA Parameters

This data structure (C) or subclass (Java) contains all the information necessary to establish a connection between an external system 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 CONNECT 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 system 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

CONNECTDATA Parameters — (1 of 3)

Parameter Name	Description	I/O	Req	Length (Bytes)
ExImMode	Indicates whether a connection is requested for: • Export (0) • Import (1) Each connection must be of either one type or the other, never both.	I	Y	1
MsgAckLvl	QOS 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 QOS 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: • 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 QOS parameter describing the level of security. The security level can be: • 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 QOS parameter that overrides middleware settings by application or document.	I	N	8

Parameter Name	Description	I/O	Req	Length (Bytes)
MsgRouting	QOS 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	QOS 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	QOS 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 QOS 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 listening for the connection, set AppHndl to the same Q/LinQ-assigned session ID displayed on the Interface Session Monitor screen. • If Q/LinQ is requesting a connection with the application, set AppHndl to session ID by default or override it with the application's value.	O	N	16

CONNECTDATA Parameters — (3 of 3)

qqReceiveConnect

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

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

See page 55.

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 Type
PortNo	Port number on the local host to be used by the application for communication with Q/LinQ.	I	N	int	int
PollWait	After Q/LinQ connects with the external system, 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 system initiates the connection, this timeout limit is not used.	I	N	int	int
CData	Q/LinQ connection parameters (see “CONNECTDATA Parameters” on page 50).	O	N	CONNECT DATA	connect Data

Return Codes

Return Code	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_LISTEN_FAIL	Listen mode failed.	2628
QQAPI_REMOTE_CONNECT_FAIL	Failed to establish a remote connection with Q/LinQ.	2634
QQAPI_CONNECT_ACK_RECV_FAIL	Failed to receive a connection acknowledgment from Q/LinQ.	2631
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.	2627

qqAckConnect

This routine answers a connection request from Q/LinQ when Q/LinQ initiates communications. It can also pass Q/LinQ a handle for addressing the external system 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 Type
ConnStat	Indicates whether or not a valid connection has been made. • 0, if connected. • Otherwise, the return code of an adapter-defined exception condition.	I	Y	int	int
AppHndl	Handle assigned by the external system to this connection or thread and used for subsequent calls.	I	N	char *	byte

Return Codes

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

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 55) and precede the sending of document data to Q/LinQ using qqSendDocData (see page 62).

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	Req	C Type	Java Type
CtrlSend	Pointer to the control data regarding the interface document being sent to Q/LinQ.	N	CTRL HEAD *	ctrl Head

Return Codes

Return Code	Description	Value
QQAPI_SUCCESS	Control data sent to Q/LinQ successfully.	0
QQAPI_SEND_DOC_CTRL_FAIL	Failed to send document control data to Q/LinQ.	2633

CTRLHEAD Parameters

This data structure (C) or subclass (Java) contains 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 is required at the beginning of every 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 CTRL in this case.	I	N	4
AppID	If the interface document is being received from Q/LinQ, this field is the system 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 system 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: • 0, document is independent, relative sequence non-significant. • 1, document is independent, relative sequence significant. • 2, document is part of a group that must be committed together. • 9, line is continued from last document.	I	Y	1
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 system and not used by Q/LinQ.	I	N	10

CTRLHEAD Parameters — (1 of 5)

Parameter Name	Description	I/O	Req	Length (Bytes)
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
DocType	Required for documents that are registered with DocType 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 system.	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 system to process the interface document.	I	N	12
AckLvlReqd	Indicates the acknowledgment level requested by the source system, either Q/LinQ or the external application. Valid values are: • 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. Passed as a reference from the sending system (such as an acknowledgment).	I	N	8

CTRLHEAD Parameters — (2 of 5)

Parameter Name	Description	I/O	Req	Length (Bytes)
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: - American, where the comma separates thousands and the period is a decimal point - European, where 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
TimeZone	The time zone of DateCreate and TimeCreate .	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 system.	I	N	10
SrcTask	Identifies the transaction in the system that caused the interface document to be generated, either MFG/PRO or the external system.	I	N	10
SrcProc	Identifies the source procedure in the system 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 system, recorded for audit trail purposes.	I	N	8

CTRLHEAD Parameters — (3 of 5)

Parameter Name	Description	I/O	Req	Length (Bytes)
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	QOS 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
MsgSync	Not implemented. Relevant only to some middleware products. The QOS 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: • 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 QOS parameter describing the level of security. The security level can be: • 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

CTRLHEAD Parameters — (4 of 5)

Parameter Name	Description	I/O	Req	Length (Bytes)
MsgPriority	QOS 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	QOS 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
MsgTTL	QOS 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	QOS 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 QOS 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 Parameters — (5 of 5)

qqSendDocData

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 56).

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 Type
CtrlBreak	2: the data represents the end of the document. 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	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 62), and precede the receiving of document data from Q/LinQ using qqReceiveDocData (see page 65).

C Syntax

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

Java Syntax

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

Parameters

Parameter Name	Description	I/O	Req	C Type	Java Type
CtrlRecv	Control data regarding the interface document being received from Q/LinQ.	O	N	CTRLHEAD	CtrlHead

Return Codes

Return Code	Description	Value
QQAPI_SUCCESS	Data received successfully.	0
QQAPI_RECV_DOC_CTRL_FAIL	Failed to receive data from Q/LinQ.	2635

CTRLHEAD Parameters

See page 57 for CTRLHEAD parameters.

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 63).

C Syntax

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

Java Syntax

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

Parameters

Parameter Name	Description	I/O	Req	C Type	Java Type
Method	The type of message sent by Q/LinQ that has been received by the adapter program.	O	N	char *	string
CtrlBreak	• 2: the data represents the end of the document. • 1: the data represents the end of the current line in the document. • 0: the data continues to the next line.	O	Y	char *	string
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	Description	Value
QQAPI_SUCCESS	Success.	0
QQAPI_RECV_DOC_DATA_FAIL	Failed to receive document data from Q/linQ.	2474

qqAckReceivedDoc

This routine sends an acknowledgment to Q/LinQ after the external system receives each packet of document data using qqReceiveDocData (see page 65). This signals Q/LinQ that the external system properly received the data 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 Type
AckValue	A QQAPI_SUCCESS indicates the external system 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	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

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

- End all Q/LinQ-initiated sessions.
- Terminate connections after a fatal error, such as the loss of data communication between the adapter and external system. When data communication is lost, the routine signals Q/LinQ that previously sent or received documents must be resent or re-received.

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 Type
ExImMode	• Use <code>qqapiunix.h</code> (UNIX) or <code>qqapiwin.h</code> (Windows) parameter <code>IMPORT</code> when disconnecting after importing documents to Q/LinQ. • Use <code>qqapiunix.h</code> or <code>qqapiwin.h</code> parameter <code>EXPORT</code> 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	Description	Value
QQAPI_SUCCESS	Successfully disconnected from Q/LinQ.	0
QQAPI_DISCONNECT_FAIL	Failed to disconnect properly from Q/LinQ.	2473

CHAPTER 3

Mapping API Reference

This chapter provides detailed reference information for Q/LinQ's mapping API.

Overview 72

Direct Database Access 72

Text File Dump/Load 84

Routing Maintenance (Main Screen)

Routing Code:	10-15000	OPERATION: 20
Standard Operation:	1030	Machine: INSPEC PER PROX-000
Description:	Machines per Op: 1	
Queue Time:	1.0	Wait Time: 0.0
Setup Time:	0.0	

Overview

The mapping API invokes custom or QAD-supplied data-transformation programs that:

- Map published documents from MFG/PRO format into the external system's format for export.
- Map imported documents from the external system's format into MFG/PRO format.

◆ See *User Guide Volume 10B: External Interfaces*.

Q/LinQ's mapping API calls the default mapping procedure defined in:

- Register External Application (36.8.1)
- Export Specification Maint (36.8.2)
- Import Specification Maint (36.8.3)

The default mapping procedure can be:

- A user-written mapping routine written in PROGRESS 4GL
- The Q/LinQ-supplied adaptor to Mercator, a commercial mapping package purchased from TSI and supported by Q/LinQ.
- An adapter to another commercial mapping package. Commercial packages require custom adapters that are built using Developer Q/LinQ.

The mapping API supports two distinct approaches for writing mapping adapter programs:

- Direct database access
- Text file dump/load

Q/LinQ's Mercator adapter uses the direct database option.

Direct Database Access

The direct database access approach lets Q/LinQ communicate with an external mapping program by directly reading and writing the document control tags and data one line at a time from the Q/LinQ queues. This approach is most useful when writing mapping programs in PROGRESS, where the mapping logic can perform input and output operations by making PROGRESS 4GL calls without the need for interim text files.

Program Decomposition of Mapping API Adapter

Figure 3.1 shows the general structure of a sample mapping API adapter that the Q/LinQ mapping API calls. In general, such programs read the raw Q/LinQ document data one line at a time from Q/LinQ (**qqGetRawDocLine**), then write the mapped data back to Q/LinQ one line at a time (**qqPutMappedDocLine**).

A single program call (**qqGetRawDocCtrl**) reads the document control tags from Q/LinQ, and another single program call (**qqPutMappedDocCtrl**) writes them back to Q/LinQ after mapping.

The entry point of the mapping program is the point where it is called by Q/LinQ; in this case, the beginning of the program. The entry point must match the 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.

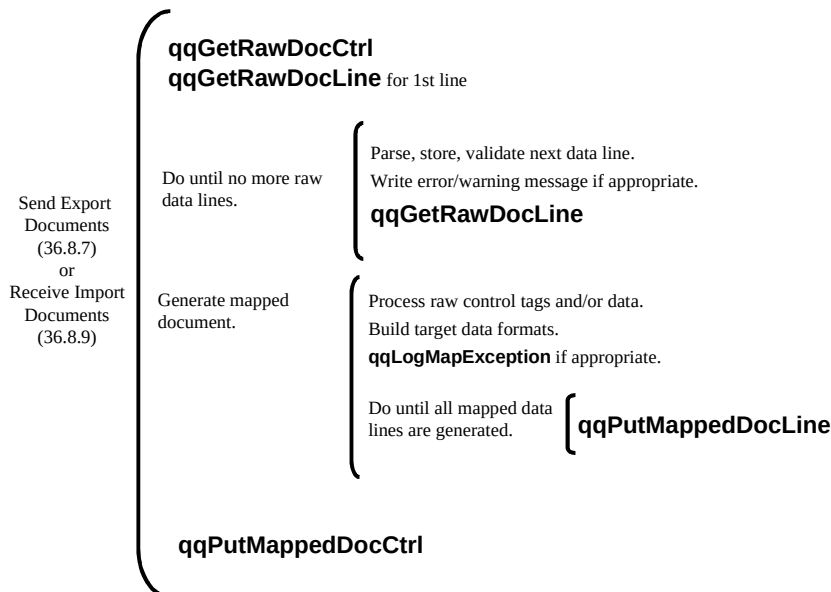


Fig. 3.1
Mapping API
Adapter Logic,
Direct Database
Access

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.
...
/* Get the raw document control tags from Q/Linq */
run qqGetRawDocCtrl
    (output AppID, output AppVer, output DocStd, output DocTyp,
     output DocRev, output
     TradPtrID, output DestProc, output SysID, output SysVer, output Src-
     Comp, output SrcTask,
     output AckLvlReqd, output DocLng, output CodePg, output DateForm,
     output NumForm, output
     SrcUserID, output SrcProc, output MfgproSite, output MfgproKey, out-
     put 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 Value3,
         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 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).
...

```

Mercator Adapter

For applications requiring complex data mapping, Q/LinQ supports one commercial mapping package, TSI Mercator. It is a powerful, general-purpose, any-to-any mapping tool. Customers license the Windows authoring tools and execution engine from TSI, not from QAD.

The Q/LinQ Mercator adapter consists of a PROGRESS persistent procedure that communicates with Mercator through a C program. The persistent procedure has a set of internal procedures, some that are defined by QAD and others specific to Mercator.

Table 3.1
Mercator Adapter
Parameters

Parameter Name	Description	Value	Default
PAGESIZE	The size of each memory page, measured in kilobytes. Maps with a large number of type references should have larger page sizes.	Multiple of 32, from 32 to 512	64
PAGECNT	The number of memory pages used.	>1	8
CTLCARD	Input card for control-tag file (*.qqc).	1 or 2	1
RAWCARD	Input card for raw data file (*.qqr).	1 or 2	2
MAPCARD	Output card for mapped tags and data file (*.qqm).	1 or 2	1
ERRCARD	Not implemented. Output card for exception message file (*.qqe).	1 or 2	2
AUDIT	Generate data audit file with unique file name for each run. All Mercator-created audit files reside in the work directory, which is defined in the qqapi.ini file.	NA	No audit
TRACE	Actuate detailed trace for all inputs and outputs. All Mercator-created trace files reside in the work directory, which is defined in the qqapi.ini file.	NA	No trace

Direct Database Mapping API

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
p_app_id	char(12)	Application ID	O
p_app_ver	char(8)	Application Version	O
p_doc_std	char(8)	Document Standard	O
p_doc_typ	char(21)	Document Type	O
p_doc_rev	char(8)	Document Revision	O
p_tradptr_id	char(8)	Trading Partner ID	O
p_dest_proc	char(12)	Destination Procedure	O
p_src_comp	char(10)	Source Component	O
p_src_task	char(10)	Source Task	O
p_ack_lvl_reqd	char(1)	Acknowledgment Level	O
p_doc_lng	char(2)	Document Language	O
p_code_pg	char(15)	Code Page	O
p_date_form	char(3)	Date Format	O
p_num_form	char(8)	Number Format	O
p_src_userid	char(8)	Source User ID	O
p_src_proc	char(12)	Source Procedure	O
p_mfgpro_site	char(8)	MFG/PRO Site	O
p_mfgpro_key	char(18)	MFG/PRO Key	O
p_ext_doc_key	char(16)	External Document Key	O
p_orig_doc_id	char(8)	Original Document ID	O
p_ctrl_tot_1	char(5)	Control Total 1	O
p_ctrl_tot_2	char(5)	Control Total 2	O
p_ctrl_tot_3	char(5)	Control Total 3	O
p_date_create	char(10)	Date Created	O

Parameter Name	Type	Description	I/O
p_time_create	char(10)	Time Created	O
p_timezone	char(10)	Time Zone	O
p_doc_sufx	char(3)	Mapped document suffix	I
Result	integer	0 upon sucessful completion, otherwise 2	I

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: • 1, at the end of the current document line • 2, end of the document • 0, neither	O

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 would typically 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
p_app_id	char(12)	Application ID	I
p_app_ver	char(8)	Application Version	I
p_doc_std	char(8)	Document Standard	I
p_doc_typ	char(21)	Document Type	I
p_doc_rev	char(8)	Document Revision	I
p_tradptr_id	char(8)	Trading Partner ID	I
p_dest_proc	char(12)	Destination Procedure	I
p_src_comp	char(10)	Source Component	I
p_src_task	char(10)	Source Task	I
p_ack_lvl_reqd	char(1)	Acknowledgment Level	I
p_doc_lng	char(2)	Document Language	I
p_code_pg	char(15)	Code Page	I
p_date_form	char(3)	Date Format	I
p_num_form	char(8)	Number Format	I
p_src_userid	char(8)	Source User ID	I
p_src_proc	char(12)	Source Procedure	I
p_mfgpro_site	char(8)	MFG/PRO Site	I
p_mfgpro_key	char(18)	MFG/PRO Key	I
p_ext_doc_key	char(16)	External Document Key	I
p_orig_doc_id	char(8)	Original Document ID	I
p_ctrl_tot_1	char(5)	Control Total 1	I
p_ctrl_tot_2	char(5)	Control Total 2	I
p_ctrl_tot_3	char(5)	Control Total 3	I

Parameter Name	Type	Description	I/O
p_date_create	char(10)	Date Created	I
p_time_create	char(10)	Time Created	I
p_timezone	char(10)	Time Zone	I
p_doc_sufx	char(3)	Mapped document suffix	O
Result	integer	0 upon sucessful completion, otherwise 2	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.	O
CtrlBreak	char(1)	Control break indicating whether the returned element is: • At the end of the current document line (1) • End of the document (2) • Neither (0)	O
Result	integer	• 0 upon successful completion, • Otherwise 2	O

qqLogMapException

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

Parameter Name	Type	Description	I/O
MsgNbr	integer	MFG/PRO message number.	I
SevLvl	integer	Message severity level. • 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	integer	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 is easier to implement than the direct database approach, 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 and the other the raw data.
- 3 The mapping program generates a mapped document file containing modified control tags and the mapped document.
- 4 The mapping API subroutine `qqLoadMappedDocFile` loads the files back into Q/LinQ and posts them to the mapped import or export queue.

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.

For the Mercator adapter, the same format used for this control-tag file is used in Mercator input card 1, the CTLCARD parameter.

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
@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	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	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).	Yes	Yes
@DOCREV	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
@SYSID	Logical identifier for Q/LinQ used in interface communications.	Yes	Yes
@SYSVER	Version of the host Q/LinQ system referenced for interface communications. Can be used to verify version compatibility or currency before processing.	Yes	Yes

Control-Tag File Format — (1 of 4)

Tag	Purpose	Exp	Imp
@DESTPROC <value>	The destination procedure. - 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: 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
@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: <code>mdy</code> , <code>dmy</code> , <code>ymd</code> .	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: - American, where the comma separates thousands and the period is a decimal point - European, where the period separates thousands and the comma is a decimal point	Yes	Yes

Control-Tag File Format — (2 of 4)

Tag	Purpose	Exp	Imp
@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. - For import documents, it links an acknowledgment document with its source document. - For export documents, the process acknowledgment uses it.	Yes	Yes
@CTRLTOT1 <value>	The optional total for multi-part documents. - 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. Can be used for batch counts and is useful for reconciling with external applications.	Yes	Yes
@CTRLTOT2 <value>	See CTRLTOT1.	Yes	Yes
@CTRLTOT3 <value>	See CTRLTOT1.	Yes	Yes

Control-Tag File Format — (3 of 4)

Tag	Purpose	Exp	Imp
@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 the Q/LinQ Control File.	Yes	Yes

Control-Tag File Format — (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.

For the Mercator adapter, the same format used for this raw-data file is used in Mercator input card 2, the `RAWCARD` parameter.

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=BUSNENTITY
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 attribute, or value, on its own line in the mapped file. Each attribute is preceded by a control tag. Q/LinQ requires the mapping program to name the mapped file with the same name as the raw data file and a qqm extension. Mercator uses the same format on output card 1, the MAPCARD parameter, when returning the mapped data to Q/LinQ.

Within each document, the tagged lines can be in any order, but the tagged lines must precede all the data lines. Q/LinQ ignores tagged 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.

See *User Guide Volume 10B: External Interfaces*.

If Mercator is the mapping program, 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)
- Export Specification Maint (36.8.2)
- Import Specification Maint (36.8.3)

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.

```
<?xml version="2.0" standalone="no"?><!DOCTYPE
CONFIRM_BOD_002 SYSTEM
"002_confirm_bod_002.dtd"><CONFIRM_BOD_002><CNTRLAR-
EA><BSR><VERB>CONFIRM</VERB><NOUN>BOD</NOUN><REVI-
SION>002</REVISION></BSR><SENDER><LOGICALID>MFG/PRO</LOGICAL
ID><COMPONENT>Q/LinQ</COMPONENT><TASK>Ack</TASK><REFEREN-
CEID>305</REFERENCEID><CONFIRMATION>0</CONFIRMATION><LAN-
GUAGE>us</LANGUAGE><CODEPAGE>iso8859-
1</CODEPAGE><AUTHID>rfj</AUTHID></SENDER><DATETIME qualifi-
er="cre-
ation"><YEAR>1999</YEAR><MONTH>01</MONTH><DAY>18</DAY><HOUR>
09</HOUR><MINUTE>28</MINUTE><SECOND>18</SECOND><SUBSEC-
OND>0000</SUBSECOND><TIMEZONE>+0200</TIMEZONE></DA-
TETIME></CNTRLAREA><DATAAREA><CONFIRM_BOD><CONFIRM><CNTRLAR
EA><BSR><VERB>SYNC</VERB><NOUN>CUSTOMER</NOUN><REVI-
SION>003</REVISION></BSR><SENDER><LOGICALID>lgs-2</LOGI-
CALID><COMPONENT>BUSNENTITY</COMPONENT><TASK>MAINTENANC</TAS
K><REFERENCEID>referenceid</REFERENCEID><CONFIRMA-
TION>1</CONFIRMATION><LANGUAGE>us</LANGUAGE><CODEP-
AGE>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><TIM-
EZONE>+0200</TIMEZONE></DATETIME></CNTRLAREA><STATUS-
LVL>99</STATUSLVL><ORIGREF>305</ORIGREF></CONFIRM><CONFIRMMS
G><DESCRIPTN>ERROR: PROCEDURE DOES NOT EXIST.</DE-
SCRIPTN><REASONCODE>1</REASONCODE></CON-
FIRMMSG></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`.

Note Mercator does not support this logging feature.

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 <code>msg_mstr</code> 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)
- Export Specification Maint (36.8.2)
- Import Specification Maint (36.8.3)

Text File Dump/Load Mapping API

The mapping adapter:

- 1 Dumps the raw Q/LinQ (`qqDumpRawDocFiles`) document tags and data to two ASCII files (`*.qqc` and `*.qqr`).
- 2 Performs the transformation.
- 3 Writes the mapped tags and data to a `*.qqm` file (`qqLoadMappedDocFile`) and the exceptions to a `*.qqe` file (`qqLoadMapExceptionFile`).

Note The parameters of each of these internal procedures are discussed in the following pages.

Program Decomposition of Mapping API Adapter

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

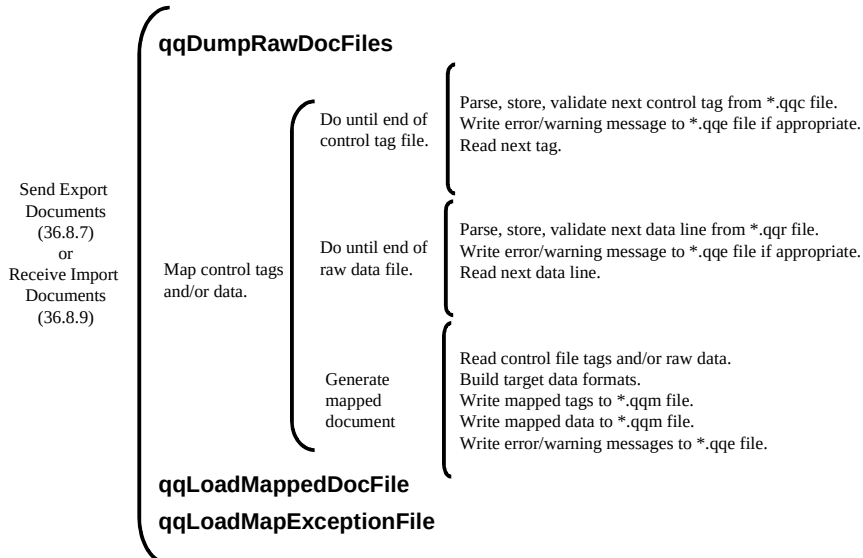


Fig. 3.5
Mapping API
Adapter Logic,
Text File
Dump/Load

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 a text file with Q/LinQ-specific tags.

Parameter Name	Type	Description	I/O
DataFileName	char(12)	Name of file that raw document data resides in.	I/O
Result	integer	0 upon successful completion, otherwise 2.	O
MappedDocCount	integer	Number of mapped documents generated from the raw document.	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	0 upon successful completion, otherwise 2.	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	0 upon successful completion, otherwise 2.	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 72.

CHAPTER 4

Publishing API Reference

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

Overview 100

Program Decomposition of Publishing API Adapter 101

14.13.2 Routing Maintenance (Main Screen)

Routing Code:	10-15000	NONAD(75) COOLING
Operation:	20	
Standard Operation		
Work Center	1030	INSPECTION, ALL SITES
Machine		
Description	INSPEC PER PROC-005	
Machines per Op	1	Milestone
Overlap Units	1	
Queue Time	1.0	
Wait Time	0.0	
Setup Time	0.0	

Route by Production Route

Overview

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

♦ See page 106 for the triplet format.

When publishing data, Q/LinQ's publishing adapter, `qqpubfwd.p`, uses an ASCII format, called a triplet format, very similar to that used in most `.ini` files.

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 module and placing them in the import queue for processing by another MFG/PRO module in the same system.

The following sample illustrates the general structure of a publishing adapter.

```
/*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" "<parameters>"}
...
/*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"
      "<parameters>"}
...
/*End repeat*/
...
/*Publish the export document to Q/LinQ's mapped or raw queues*/
if return-value = 0 then
```

♦ See parameter list on page 106.

♦ See parameter list on page 106.

```
{gprunp.i "qqpubfwd" "p" "qqPublishInterfaceDoc"
"(<parameters>)"}
```

...

See parameter list on page 110.

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 110). This reinitializes all the temporary work tables created by the program without publishing any data. Then rerun the program from the beginning.

Program Decomposition of Publishing API Adapter

The three publishing and forwarding routines are internal procedures of the persistent procedure `qqpubfwd.p`.

Q/LinQ maintains queues for raw and mapped data. `qqpubfwd.p` places data in these queues for later publishing or forwarding.

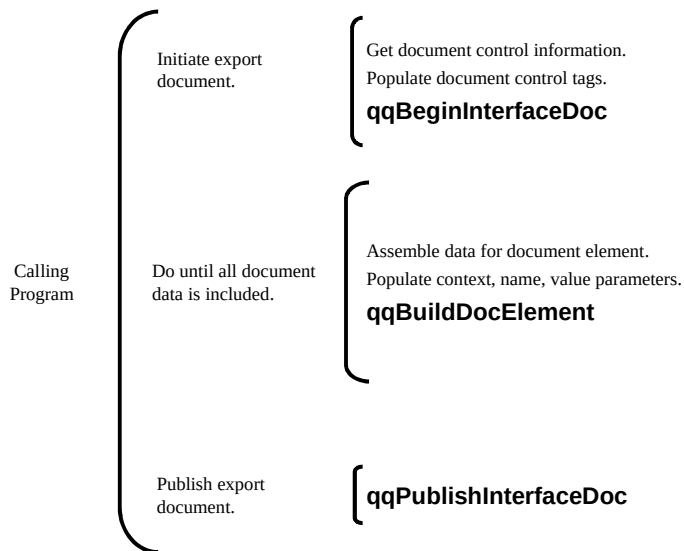


Fig. 4.1
Publishing API
Adapter Logic

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.

▶ See *User Guide Volume 10B: External Interfaces*.

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 Q/LinQ's 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
DestAppId	x(12)	Identifies the Q/LinQ registered application to which the document is to be exported. If the System ID from the Q/LinQ Control File is entered here, then the document is forwarded as an import back into Q/LinQ.	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

qqBeginInterfaceDoc — (1 of 4)

Parameter Name	Type	Description	I/O	Req
DocType	x(16)	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
DocRev	x(8)	Required for documents that are registered with DocRev values. Revision number of the exported document after being mapped by Q/LinQ.	I	Y
UnitOfWork	x(1)	Indicates how a group of documents exported together should be treated. Possible values are: • No document sequencing or grouping (0) • Documents sequenced but processed independently (1) • Documents sequenced and processed as a group (2) 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
TradPtr	x(32)	Identifies the trading partner 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

qqBeginInterfaceDoc — (2 of 4)

Parameter Name	Type	Description	I/O	Req
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: • 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

qqBeginInterfaceDoc — (3 of 4)

Parameter Name	Type	Description	I/O	Req
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 version of the source or destination application that the document came from or goes to.	I	N
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: • 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 — (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.
- `LineBreak` tells Q/LinQ to insert a line break. If `LineBreak` is absent, Q/LinQ inserts a comma. Use `LineBreak` between each document element unless you want to generate a comma-separated-value (CSV) data stream. A CSV 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.

The `Context` and `FldName` 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 these optional parameters, `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 is:

```
pt_mstr.pt_part=01-10000
```

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

```
[Line3]sod_det.sod_part=01-10000
```

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

```
ITEMNO=01-10000
```

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

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

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

- .ini files


```
[context]
FldName=FldValue
FldName=FldValue
FldName=FldValue, FldValue
...
[END]
```
- Comma-separated values (CSVs)


```
FldValue,FldValue,FldValue...
```
- Comma-separated tagged values


```
FldName=FldValue,FldName=FldValue...
```

◆ See Figure 3.3 on page 90.

When publishing MFG/PRO data that will be mapped into OAGIS BODs, use the `.ini` file format. 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.

If the data type, segment, and field names are OAGIS-defined names, they can directly map to the corresponding 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 comma separates the triplet from the next one.

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. The program ignores the undefined values.

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
Context	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, Context should be set to the data type name that it is a member of, such as SOHEADER or SOLINE in the SYNC_SALESORDER BOD.	I	Y
FldName	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 Context is also used. For OAGIS BODs, FldName should be set to the field name of the data inside the BOD; for example, SALESORDERID.	I	Y
FldValue	NA	Data value of the field identified in FldName . Q/LinQ accepts field values of any length.	I	Y
LineBreak	logical	If Yes, break line here and start a new document line for the data that follows. If No (default), append a comma to the published line and continue.	I	Y
CtrlTotalEntry1	integer	User-defined entry for Control Total 1 accumulated by Q/LinQ.	I	N
CtrlTotalEntry2	decimal	User-defined entry for Control Total 2 accumulated by Q/LinQ.	I	N
CtrlTotalEntry3	decimal	User-defined entry for Control Total 3 accumulated by Q/LinQ.	I	N

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.

The following table lists parameters of qqPublishInterfaceDoc.

Parameter Name	Type	Description	I/O	Req
CancelFlg	logical	Yes cancels the publish operation by reinitializing the temporary work tables.	I	N
CtrlTotal1	integer	Control total 1 accumulated by Q/LinQ.	O	N
CtrlTotal2	decimal	Control total 2 accumulated by Q/LinQ.	O	N
CtrlTotal3	decimal	Control total 3 accumulated by Q/LinQ.	O	N

CHAPTER 5

Processing API Reference

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

Overview 112

Gateway Programs 112

varom91.p 14.13.2 Routing Maintenance (Date Based)

Routing Code:	10-15000	NONAD(75) COOLING
Operation:	20	
Standard Operation		
Work Center	1030	INSPECTION, ALL SITES
Machine		
Description	INSPEC PER PROG-005	
Machines per Op	1	Milestone P
Overlap Units	1	
Queue Time	1.0	
Wait Time	0.0	
Setup Time	0.0	

Route by Production Route

Overview

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

♦ See *User Guide, Volume 10B: External Interfaces* for information on UI emulation.

This chapter only discusses loading with gateway programs. UI emulation is part of Runtime Q/LinQ and discussed with it.

Gateway Programs

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

When you register a gateway program, set the Process Through User Interface flag to No in Register External Application (36.8.1) and Import Specification Maint (36.8.3).

Q/LinQ's 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 system associated with the gateway program.

Q/LinQ then passes control information to the gateway through input parameters.

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 of Processing API Adapter

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

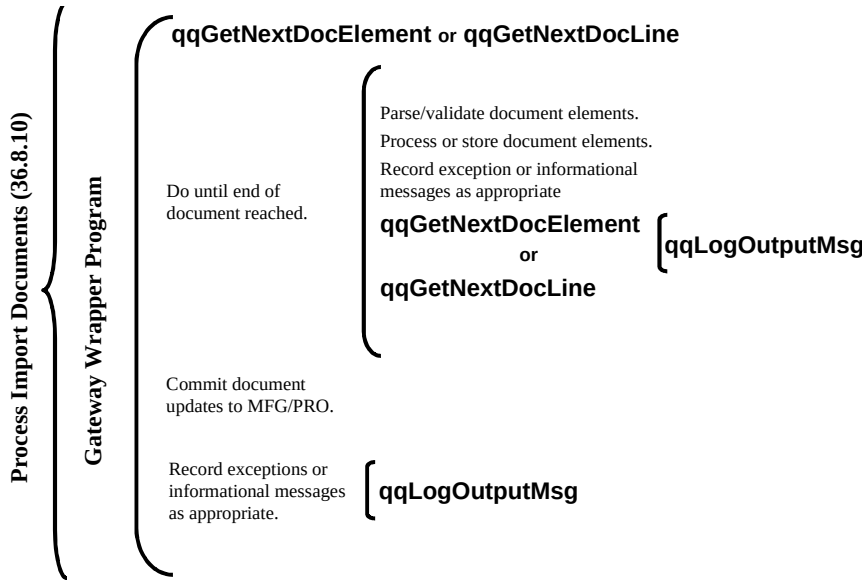


Fig. 5.1
Processing API
Gateway Program
Logic

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 (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 qqLogOutputMsg in CallingProc
    (input MsgNbr, input SevLvl, input Var1, input Var2, input Var3).
  ...
/*Get next document element from Q/LinQ*/
  run qqGetNextDocElement (output EOLDelimitOnly, output Context,
    output FieldName, output FieldValue, output CtrlBreak, output
    ResultStatus).
  ...
/*End repeat*/
  
```

Gateway Parameters

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
DocType	x(16)	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 processes the documents.	I	Y
CallingProc	handle	Handle to call processing API methods.	I	Y
ReturnStat	x(1)	Indicates the success or failure of the processing. Valid values are: • 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
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

Gateway Parameters — (1 of 2)

Parameter Name	Type	Description	I/O	Req
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 system 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 Parameters — (2 of 2)

Gateway Procedures

The gateway wrapper program makes use of three internal procedures:

- qqGetNextDocElement
- qqGetNextDocLine
- qqLogOutputMsg

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 commas or on separate lines. 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
<code>EOLDelimitOnly</code>	logical	Indicates whether the returned document element field value should be assumed: - To terminate only at the end of the current document line (Yes) - To also be terminated by a comma (No) Setting this field to Yes lets the routine parse non-CSV data values.	I	N
<code>Context</code>	x(32)	Context (first token) of the next document element or triplet. Can be null if the element contains no context token.	O	N
<code>FieldName</code>	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
<code>FieldValue</code>	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 comma characters only if <code>EOLDelimitOnly</code> is Yes.	O	N

Parameter Name	Type	Description	I/O	Req
CtrlBreak	x(1)	Control break indicating whether the returned element is: • 1: at the end of the current document line • 2: end of the document • 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
LineCont	logical	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	Y

qqLogOutputMsg

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: • 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

CHAPTER 6

Q/LinQ Utilities

This chapter provides reference information on Q/LinQ utilities.

Check Registry 122

Identify Source Application 123

Routing Maintenance (Date Based)

Routing Code: 10-15000

Description: INSPEC PER PROC-005

Standard Operation: 10:30

Work Center: 10:30

Machines per Op: 1

Overlap Units: 1

Queue Time: 1.0

Wait Time: 0.0

Setup Time: 0.0

Check Registry

Use `mgckapp.p` to:

- Determine whether an external application has been registered in Register External Application (36.8.1)
- 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	character	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	character	Description of the application.	O	N
AppName	character	Name of the application.	O	N
AppVer	character	Version of the application.	O	N

Identify Source Application

Use `mgqqapp.i` 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 Q/LinQ's user-interface emulation mode. It does not affect the MFG/PRO program's processing.

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"}
```


Q/LinQ 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 adapted program’s services 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)*.

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 an application's services 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*.

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.

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.

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.

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 Q/LinQ's 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.

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 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.

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)*.

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*.

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*.

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 the initiating application's

execution can continue. 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)*.

Index

A

- acknowledgment
 - Q/LinQ communications 14
 - qqAckReceivedDoc 41
- adapters
 - export 13
 - import 13
 - mapping 75
 - Mercator 75, 76
 - messaging API 38
 - MQSeries 15
 - publishing 100
 - stream API 16
- application program interface (API)
 - mapping 72
 - message-oriented middleware (MOM) 15
 - messaging 37
 - processing 111
 - publishing 100
 - stream 16

C

- comma-separated value (CSV) 106
- communications APIs
 - message-oriented middleware (MOM) 15
 - messaging 37
 - overview 12
 - stream API 16
- CONFIRM_BOD BOD
 - control-tag file 88
- CONNECTDATA parameters 50
- control tags 84
- CTRLHEAD parameters 57

D

- dump/load
 - text file 84

E

- errors
 - mapping API 92
 - mapping API text file dump/load 97
 - processing API 119
- Export Specification Maint
 - input/output card parameters 91
 - mapping API 72

F

- file formats
 - control tag for mapping API 84
 - error data for mapping API 92
 - mapped data for mapping API 90
 - raw data for mapping API 89

G

- gateway programs 112
 - parameters 114
 - qqGetNextDocElement 116
 - qqGetNextDocLine 118
 - qqLogOutputMsg 119

I

- Import Specification Maint
 - gateway programs 112
 - input/output card parameters 91
 - mapping API 72
- ini file format
 - publishing API 107
- IP address
 - Q/LinQ use of 14

M

- mapping API
 - adapter decomposition 73
 - direct database access 72
 - qqGetRawDocCtrl 77

- qqGetRawDocLine 79
- qqLogMapException 83
- qqPutMappedDocCtrl 80
- qqPutMappedDocLine 82
- Mercator 75
- overview 72
- text file dump/load 84
 - CONFIRM_BOD BOD 88
 - control tags 84
 - error file format 92
 - mapped-data file format 90
 - qqLoadMapExceptionFile 97
 - qqLoadMappedDocFile 96
 - raw-data file format 89
- Mercator 75
- message-oriented middleware (MOM) API
 - overview 15
- messaging API 37
 - adapter decomposition 37
 - CONNECTDATA 50
 - CTRLHEAD 57
 - functional description 41
 - limitations 15
 - overview 13
 - program decomposition 42
 - qqAckConnect 55
 - qqAckReceivedDoc 41, 67
 - QqApi 46
 - qqDisconnect 68
 - qqReceive Connect 53
 - qqReceiveConnect 53
 - qqReceiveDocCtrl 63
 - qqReceiveDocData 41, 65
 - qqSendConnect 48
 - qqSendDocCtrl 56
 - qqSendDocData 41, 62
 - return codes 43
 - routine libraries 38
 - security issues 15
 - testing communications 44
- MOM. *See* message-oriented middleware (MOM) API
- MQSeries 15

O

- OAGIS BOD
 - CONFIRM_BOD 88
 - mapping 85
- OAGIS BODs
 - publishing to 108

P

- Process Import Documents

- gateway programs 112
- processing API 111
 - adapter decomposition 113
 - adapter example 113
 - gateway programs 112
 - parameters 114
 - qqGetNextDocElement 116
 - qqGetNextDocLine 118
 - qqLogOutputMsg 119
- process-to-process communications 13
- PROGRESS
 - mapping adapter 75
 - mapping programs 72
 - stream API use of 16
 - use with publishing adapter 100
- publishing API 100
 - adapter decomposition 101
 - adapter example 100
 - OAGIS BODs 108
 - qqBeginInterfaceDoc 102
 - qqBuildDocElement 106
 - qqpubfwd.p 101
 - qqPublishInterfaceDoc 110
 - triplet format 100

Q

- Q/LinQ Control File
 - System ID value 102
 - time zone 88
- qqAckConnect 41, 55
- qqAckReceivedDoc 67
- QqApi
 - constructor 46
 - parameters 46
 - public class Logger 46
- QqApi Java class 46
- qqBeginInterfaceDoc 102
- qqBuildDocElement 106
- qqCloseSrcDest 36
- qqDisconnect 41, 68
- qqGetNextDocElement 116
- qqGetNextDocLine 118
- qqGetRawDocCtrl 77
- qqGetRawDocLine 79
- qqLoadMapExceptionFile 97
- qqLoadMappedDocFile 96
- qqLogMapException 83
- qqLogOutputMsg 119
- qqOpenSrcDest 19
- qqpubfwd.p 101
- qqPublishInterfaceDoc 110
- qqPutMappedDocCtrl 80

- qqPutMappedDocLine 82
- qqReadDocCtrl 30
- qqReadDocData 35
- qqReceiveConnect 41, 53
- qqReceiveDocCtrl 41, 63
- qqReceiveDocData 65
- qqrecv adapter 38, 44
- qqsend adapter 38, 44
- qqSendConnect 41, 48
- qqSendDocCtrl 56
- qqSendDocData 62
- qqWriteDocCtrl 22
- qqWriteDocData 29

R

- Register External Application
 - checking registered applications 122
 - DocRev value 23, 31
 - DocStd value 23, 31
 - DocType value 23, 31
 - gateway programs 112
 - input/output card parameters 91
 - mapping API 72
- return codes 43

S

- security of messaging API 15
- sock_stream 14
- stream API
 - adapter decomposition 16
 - overview 13
 - qqCloseSrcDest 36
 - qqOpenSrcDest 19
 - qqReadDocCtrl 30
 - qqReadDocData 35
 - qqWriteDocCtrl 22
 - qqWriteDocData 29

T

- TCP sockets
 - Q/LinQ use of 14
- testing communications
 - qqrecv 44
 - qqsend 44
- text file dump/load 84
 - control tags 84
- triplet format 100
- TSI Mercator 75

