Industry-specific

QAD SOLUTIONS

Manufacturing Applications

Technical Reference
QXtend Inbound

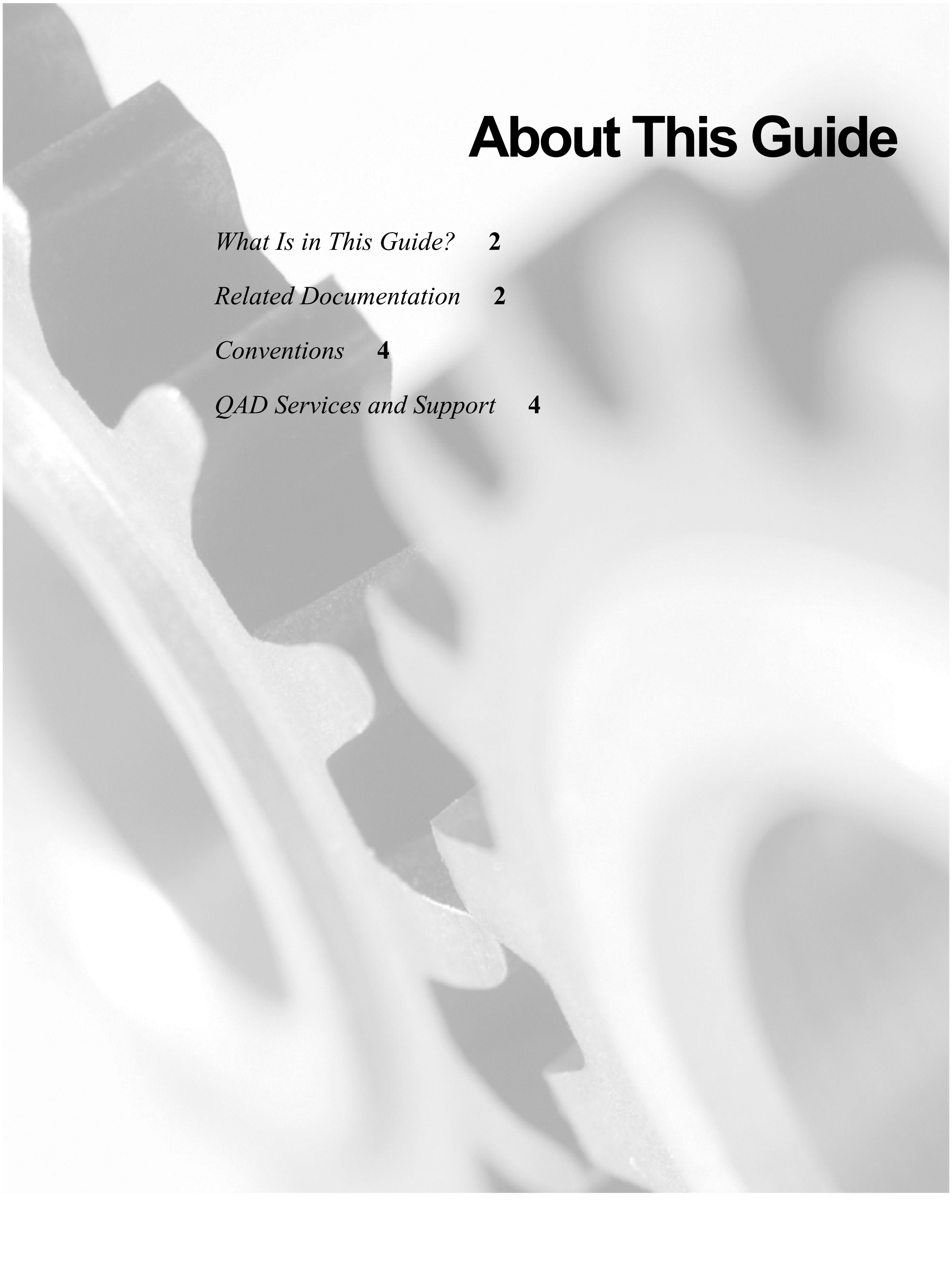
Overview and Installation
User Information
Implementation
Reference

QAD

78-0602F
MFG/PRO eB, eB2, and eB2.1
QXtend Inbound 1.2.1.6
May 2006

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

Contents



About This Guide

What Is in This Guide?

Use this guide to install, configure, and use QXtend Inbound (QXI). The guide is divided into the following sections:

Section	Page
Overview and Installation	page 7
Configuring and Using QXtend Inbound	page 45
Additional Implementation Options	page 109
Reference	page 153

Audience

These instructions are intended for a system administrator with experience installing MFG/PRO, as well as configuring and managing hardware and operating system software. This person also should have a good understanding of networking concepts and administration, as well as an understanding of concepts and technologies discussed in "QXtend Inbound Overview" on page 10.

If you do not have this expertise within your company, contact QAD Support for information on the installation and customization offerings supplied by QAD's Global Services.

Installation and Product Updates

Check the QAD Web site to make sure you have the latest installation errata, installation guides, and installation media. You can also access revised or new QDocs.

http://support.qad.com/

Related Documentation

Progress Documentation

- For information on installing and configuring the Progress AppServer, see *Building Distributed Applications Using the Progress AppServer.*

You can find the complete Progress documentation set online at:

http://www.progress.com/products/documentation/index.ssp

Related QAD Documentation

- For information on installing and setting up QXtend Outbound, refer to *Technical Reference: QXtend Outbound.*
- For information on installing MFG/PRO or converting to a more recent release, refer to the appropriate installation guide for your system or the appropriate MFG/PRO conversion guide.
- For information on using MFG/PRO functions, refer to the *User Guides*.
- For information on using Q/LinQ, see *External Interface Guide: Q/LinQ* and *Technical Reference: Q/LinQ*.
- For information on installing and configuring QAD JIT Sequencing, refer to *Installation Guide: QAD JIT Sequencing.*
- For information on using QAD Production Scheduler, see *Technical Reference: QAD Production Scheduler.*

For QAD customers with a Web account, the complete MFG/PRO documentation is available for review or downloading at:

http://support.qad.com/

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

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

Conventions

Menu References

This guide applies to MFG/PRO eB, eB2, and eB2.1. A number of menus have been reorganized between these releases. Differences in menu numbers are noted, when necessary, using the following convention:

User Tool Maintenance (36.20.4; 36.20.2 in eB)

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

Typographic Conventions

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

If you see:	**It means:**
`monospaced text`	A command, path, or file name.
`italicized monospaced text`	A variable name for a value you enter as part of an operating system command; for example, *`YourCDROMDir`*.
`<italicized monospaced text>`	A variable for a system value such as a drive letter or machine name; for example *`<hostmachine>`*.
Note	Alerts the reader to exceptions or special conditions.
Important	Alerts the reader to critical information.
Warning	Used in situations where you can overwrite or corrupt data, unless you follow the instructions.

QAD Services and Support

During your implementation of QAD products, you should be aware of available service and support offerings.

QAD Consulting and Technical Services

MFG/PRO installations have a wide variety of configuration possibilities, are highly scalable, and are easily customized. While this guide provides basic installation information, it cannot consider every possible computing environment or configuration.

To take full advantage of MFG/PRO's flexibility and potential in your specific environment, QAD offers Consulting and Technical Services specializing in installations and customizations. These offerings include performance enhancements as well as technical and administration training. For more information, contact your nearest QAD office or go to:

http://www.qad.com/services/

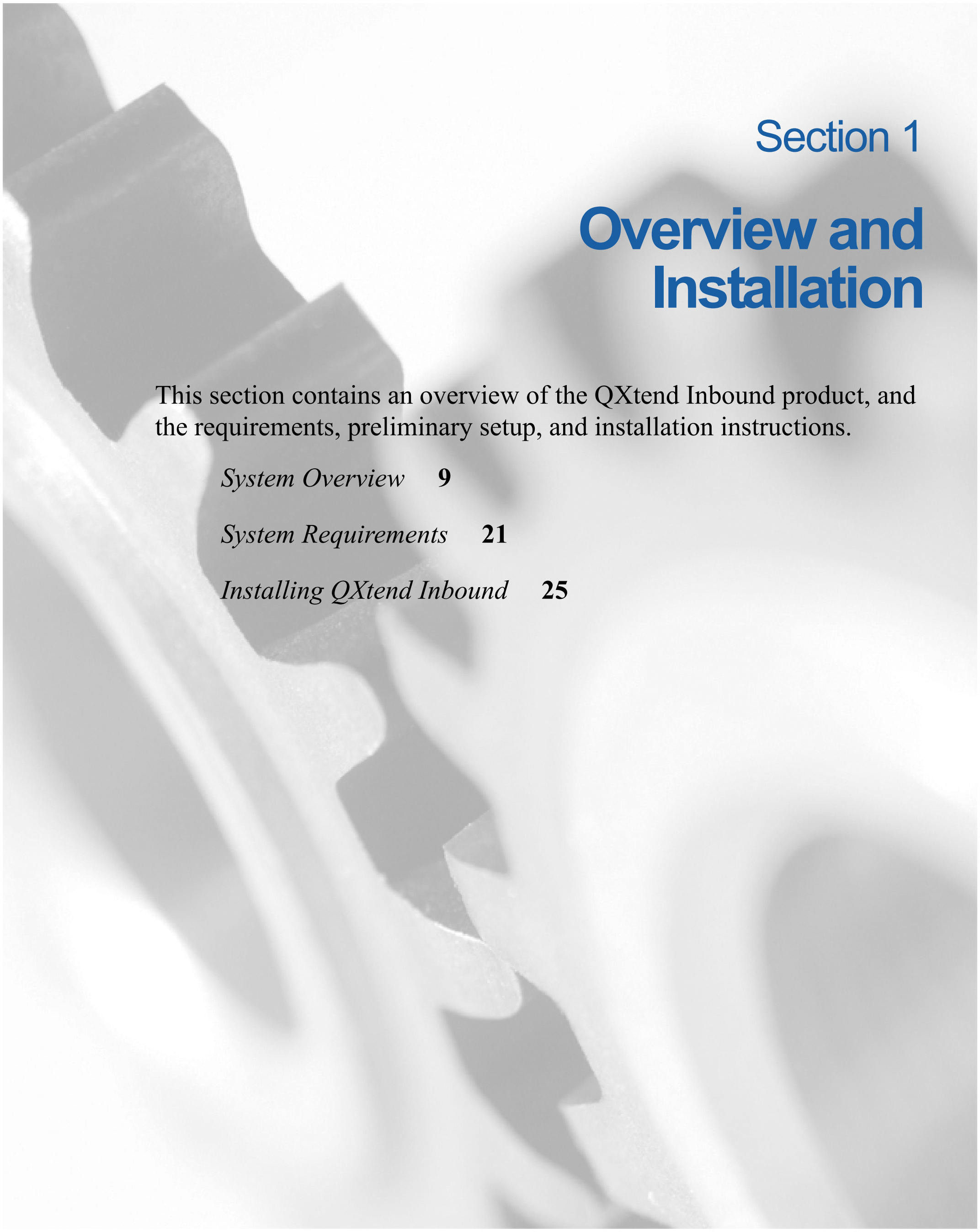
QAD Support

Before beginning an installation, it is recommended that you contact QAD Support to notify them of your plans, schedule, and if you will have a consultant onsite to assist in the process. Be sure to identify who the consultant is, and how they can be reached. QAD Support provides a number of online resources to help you find answers to your questions and avoid problems beforehand. These include, but are not limited to, the following:

- QAD Knowledgebase
- QAD Learning Portal
- Customer Advisories
- Platform Availability Guide
- QVillage Discussion Forums

Go to QAD ServiceLinQ to access the full array of online Support resources at:

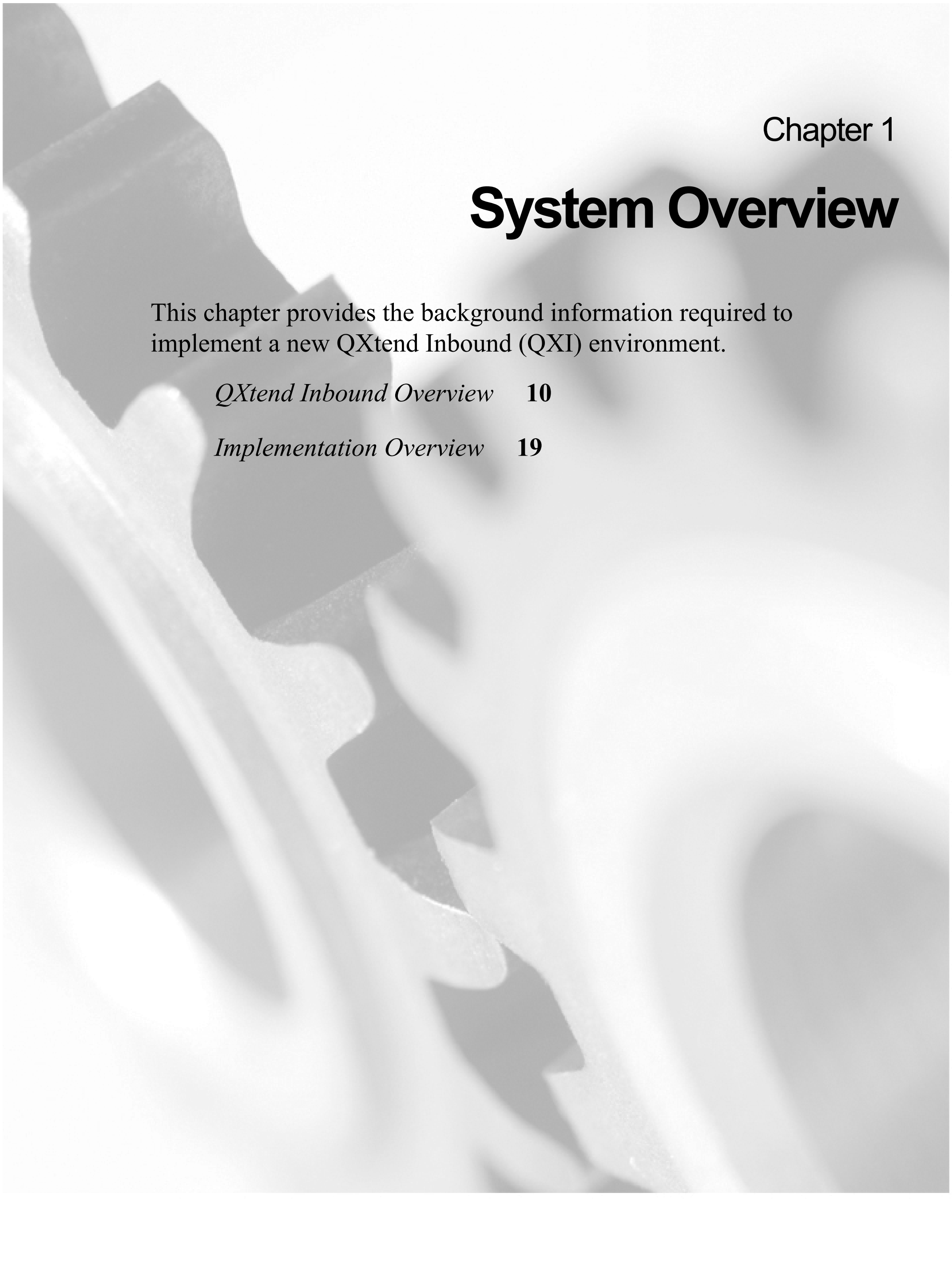
http://support.qad.com/



Section 1

Overview and Installation

This section contains an overview of the QXtend Inbound product, and the requirements, preliminary setup, and installation instructions.



Chapter 1

System Overview

This chapter provides the background information required to implement a new QXtend Inbound (QXI) environment.

QXtend Inbound Overview

See "Glossary" on page 265 for definitions of terms and abbreviations.

QXtend Inbound (QXI) is a component of the QAD QXtend interoperability framework. This framework provides a standardized data interface between QAD products, and between QAD products and external systems. The interface is a Web services-based, SOAP-compliant, XML framework, enabling complete platform-independent access to MFG/PRO business functionality.

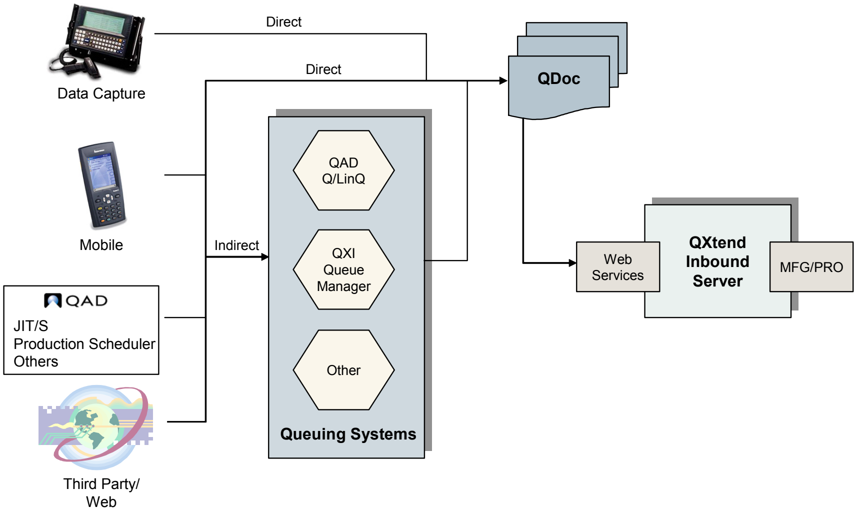
The framework consists of:

- QXtend Inbound, which imports SOAP-compliant XML QDocs from external applications into MFG/PRO and other QAD applications such as QAD JIT Sequencing (JIT/S) and QAD Production Scheduler
- QXtend Outbound, which exports user-configurable business objects as XML QDocs out of QAD products such as MFG/PRO

See *Technical Reference: QXtend Outbound* for details.

QXtend Inbound delivers inbound data to MFG/PRO eB, eB2, or eB2.1 menu-level programs and to a limited number of QAD JIT/S data tables. The inbound data is in the form of XML data documents, called QDocs, delivered in either a proprietary XML format or, for a select few programs, as direct code APIs. Figure 1.1 illustrates the QXI system architecture.

Fig. 1.1
Basic QXI System Architecture



As shown in Figure 1.1, incoming documents—called requests—can be generated directly from data capture tools such as barcode readers; mobile devices; other QAD products such as Just-In-Time Sequencing, QAD Production Scheduler, EDI ECommerce, or Trade Management; and from any third-party application that can generate XML.

The data is formatted as SOAP-compliant XML documents using QDoc templates in XML schema format. These documents are fed directly to QXtend Inbound. Alternatively, you can queue the documents using a number of queue management interfaces, including the QXI Queue Manager and QAD's Q/LinQ product.

See Chapter 11, "Using QXtend with Q/LinQ," on page 143 for more details.

In addition, you can queue XML documents that do not meet QXI standards, and call the QXtend Inbound transformation engine to apply custom XSLT style sheet transformations to the document. The transformation engine outputs a valid QDoc request.

"Configuring the Transformation Engine" on page 65 has more details.

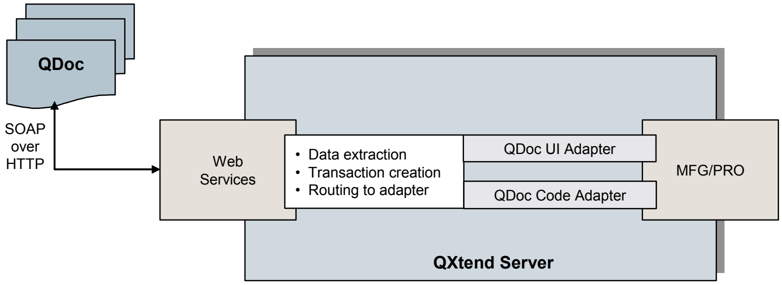
QXtend Inbound accepts the XML QDoc requests as Web service calls as shown in Figure 1.2. Optionally, you can send the incoming QDocs through the QXI preprocessor to validate the incoming data, eliminating incorrect QDocs. You can also create postprocessing programs to validate response data.

See Chapter 9, "Using QXtend Pre- and Postprocessors," on page 129 for details.

For all QDocs, the QXI server validates the user ID and password from the SOAP envelope if you have configured it to do so. It then extracts the MFG/PRO data from the SOAP envelope, repackages the data in a transaction container, and then sends the data to an MFG/PRO session, called a receiver, through either the UI adapter or the code adapter.

See "Implementation Overview" on page 19 for additional information.

Fig. 1.2 QXtend Inbound Architecture



When MFG/PRO receives the data from the UI adapter, it enters it through a telnet user session with a menu-level business function. The data is validated against all database, menu-level, and program validations, and all database triggers are fired.

Data received from the code adapter is broken into temporary tables that are sent to MFG/PRO through direct calls to Progress API programs. This process is much faster, but requires an extensive API.

Once the data has been input, the adapter generates a response document that is returned through QXI in the same transaction container that delivered the request. The response document provides a success or failure notification. The failure notification can contain warnings and errors. The response document can also contain data from MFG/PRO such as the primary data key value and additional error information, such as Java trace data or pre- or postprocessing errors.

Direct API Programs

The direct code APIs address programs that require the fastest possible data transfer into MFG/PRO. The code APIs use the native MFG/PRO online application processing programs, but without executing the user interface. The inbound data format for code APIs is XML documents in QAD-proprietary format.

The code APIs shipping with QXtend Inbound are:

- maintainSupplier (`advnai01.p`)
- maintainPurchaseOrder (`popoai01.p`) for both discrete and scheduled orders
- maintainSupplierSchedule (`rssai01.p`)
- receiptsBackwardExplode (`icunrc01.p`)

Note To use code APIs you must set up a Progress AppServer. See Chapter 10, “Configuring the Progress AppServer,” on page 135 for additional information on AppServer setup.

QDocs

Each inbound XML QDoc request results in an XML QDoc response. Each QDoc references the MFG/PRO session and version and the target MFG/PRO application program. A QDoc can contain one or more records for that calling program, but cannot call other programs in MFG/PRO.

See Chapter 13, "QDoc Structure Reference," on page 165 for examples of QDoc files.

Most incoming QDoc requests are sent to MFG/PRO over a telnet session. The target calling program is launched and the data from the QDoc passed down field-by-field just as if a user were entering the data. If data is missing for a non-mandatory field, the session skips through the field, always matching data input with the correct field in the calling program interface.

To make this screen-driven entry possible, each QDoc has matching XML schemas and events documents stored on your QXI drive.

QDoc XML Schemas

In most cases, two XML schema files are associated with each supported QDoc request and its MFG/PRO calling program. All standard XML schemas are available on the Web by clicking the QDoc link under Downloads on the QAD Support Web site at:

See Chapter 15, "QDoc Specifications and Standards," on page 193 for details.

http://support.qad.com/

The XML schemas for a calling program such as Sales Order Maintenance (`sosomt.p`) contain all possible data entry fields in the calling program. They also include the information required to start data iterations, which are repetitive entry sequences such as sales order lines.

For information on schema exceptions, see "QDoc Exceptions" on page 253.

For example, the schema documents for `sosomt.p` in eB2 are:

- `maintainSalesOrder-eB2_1.xsd`
- `salesOrderType-eB2_1.xsd`

The first file is the base schema file, and contains calling procedure information. The second file, the type file, contains the detailed field listing.

For additional information on configuring response document contents, see "Controlling QDoc Response Data" on page 48.

Two XML schema files are also provided for each QDoc response. They are named similarly to the request QDoc schemas, but include `response` in the name. For `sosomt.p` in eB2, the response schema files are:

- `maintainSalesOrderResponse-eB2_1.xsd`
- `salesOrderResponseType-eB2_1.xsd`

The first file is the base schema file, and contains calling procedure information. The second file, the type file, contains a list of primary key fields for each iteration. Editing the type file enables you to filter out or add MFG/PRO field values you do not want to see in the response document.

QDoc Events Documents

For information on event exceptions and errors, see "Event Exceptions" on page 245.

The QDoc also needs information on how to navigate in an MFG/PRO screen. By default, the QDoc sends data and moves to the next field. However, it needs to know, given the field location, when to accept an iteration—as for multiple sales order lines within a single sales order—and when to allow a deletion. These screen actions are called events and are stored in a version-specific directory for each calling program in an events document named by default after the calling program. The default events document for `sosomt.p` is `sosomt.xml`. The base events documents are also available on the QAD Web site.

You can create additional events files to support multiple processing scenarios within the same calling program such as for implementations that use European Accounting for some, but not all, transactions. Multiple events files can also be used to test processing paths to improve performance.

Custom QDocs

See Chapter 8, "Using QGen," on page 111.

You can create QDoc XML schema and events documents for programs you have created or customized using a tool called QGen that ships with QXtend Inbound. QGen creates schema files for request and response documents, as well as an events file.

To create the supporting files, you launch an MFG/PRO QGen session and run the calling program you want to create QDocs for. When it is running, you start the QGen mapping functionality. You then navigate through each field in the program. For each field, a pop-up dialog appears, letting you enter schema and events information for the field. When you have completed the mapping for the program, you save the QGen map and turn off the mapper.

From the same calling program, you then open the QGen menu and load the mapping data. You then enter the QDoc name, for example, `maintainCustomData`, and generate the documents.

To create QDocs for customized MFG/PRO programs, you simply reload the existing mapping program for the original calling program. You can then resave it under a new name—`sosomt.p` to `xxsosomt.p`—and then map the new or modified fields in the program.

Transforming Non-Standard XML Documents

Typically, the QXI Queue Manager expects all incoming documents to be in proprietary QDoc format. The Queue Manager can add a valid SOAP envelope if one is missing. It can also accept non-standard XML documents and transform them into QDocs with a valid SOAP envelope using the QXI transformation engine.

▶ "Configuring the Transformation Engine" on page 65 has more details.

The transformation engine is triggered when an XML document arrives with an extension other than `.req`. Any non-QDoc XML document is picked up by the transformation engine. Depending on the file extension, an XSLT style sheet is applied to the document to generate a valid QDoc, a SOAP envelope is added, and the new QDoc is sent to the correct Queue Manager directory with an `.req` extension.

The transformation engine ensures all the linkages between directories, file names, and processing flows. Individual companies must provide their own XSLT style sheet transformations for the transformation engine to use.

WSDL Document for Web Service Clients

In order to directly call QXtend Inbound as a Web service, a client component must be developed that is capable of calling QXI using the SOAP and HTTP protocols. Web clients can be written in many programming languages, but most are developed with the help of software development tools or frameworks that generate much of the low-level code responsible for managing the details of SOAP and HTTP communications. To generate the code, most Web service tools require a special XML document written in the Web Services Description Language (WSDL).

WSDL is a proposed standard intended to precisely describe all the technical information needed by an external process to call the interface of a specific Web service—its supported methods or operations, the parameter names and data types required by the operations, the network ports where the Web service listens for requests, and so on.

See “Modify the WSDL Document” on page 39 for instructions.

QXtend Inbound provides a single WSDL document that you can modify manually to designate the network address (URL) in the QXtend Inbound installation where the Web service will run.

Directory Structure

When you install QXtend Inbound, two separate steps install files:

- Java code is placed under the Tomcat installation directory.
- Progress code is typically placed under your MFG/PRO installation directory, by default `mfgsvr`.

Java Files

The QXI Tomcat directory is located in `*TOMCAT_HOME*/webapps/qxtendserver`. Figure 1.3 on page 17 illustrates the extended structure. A brief description of the main directory nodes follows.

descriptors. This directory contains release-specific subdirectories that each contain two XML files listing all release-specific QDoc schemas for your system. The list of standard QDocs released by QAD is maintained in `qadQdocs.xml`. A list of all custom QDoc schemas you have created on your system is stored in `implementationQdocs.xml`.

events. This directory contains release-specific XML files that describe iteration and non-standard navigation events for each supported calling procedure. A `globaldefaults.xml` document at the top level contains the default events that all QDocs rely on.

receivers. This directory is specific to your implementation. You create receivers for each MFG/PRO session and set of QDocs. Those receivers are added to the `qdocReceivers.xml` file at the top level of this directory. Each receiver is also given a subdirectory under `/receivers`. In each receiver subdirectory are a `/descriptors` subdirectory and a `/requests` subdirectory.

- The `/descriptors` subdirectory, like the global one described previously, contains a `qadQdocs.xml` that lists standard QDocs you have assigned to this receiver. It also contains an `implementationQdocs.xml` that lists all custom and other non-QAD QDoc schemas you have assigned to this receiver.
- The `/requests` subdirectory is a location where you place sample QDocs for testing purposes. When you use a test page, the file name you enter must exist in this directory.

schemas. This directory contains release-specific XML schema files for each supported calling procedure.

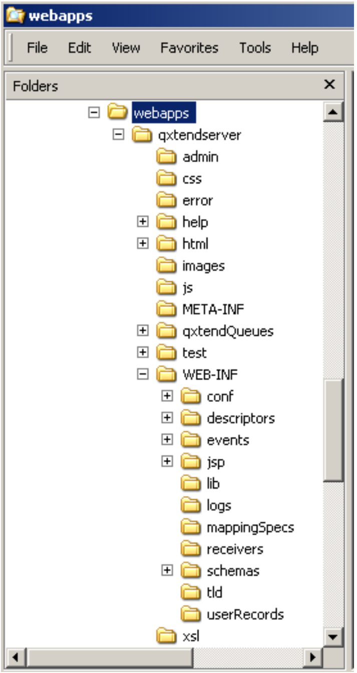


Fig. 1.3
QXtend Inbound Directory Structure

Progress Code

The QXI Progress files can be installed anywhere on the QXtend Inbound machine, but are typically installed under the MFG/PRO installation directory into a directory called `qxtendserver`; for example, `c:\mfgsvr\qxtendserver`.

The installation script copies the correct service-pack specific files based on your version of MFG/PRO, creating the following directories:

qxxrc. This directory contains the QXI Progress source code.

qxdata. For MFG/PRO eB installations only, this directory contains data files to be loaded into the MFG/PRO admin database.

tools. This directory contains the QGen utility and its data files.

tools/datFiles. These are the release-specific master data files created by the QGen tool during the mapping of a calling program. These files can be loaded into QGen in order to generate new schema and events documents.

Additional data files and the compilation file, `qxtendcompile.wrk`, are extracted into the root `qxtendserver` directory.

The MFG/PRO installation directory may also be updated with MFG/UTIL files, if they are needed for the current installation.

Supported QXtend Integrations

QXtend Inbound is either required or available for certain QAD products. These include QAD Just-In-Time Sequencing (JIT/S), QAD Production Scheduler, and QAD Mobile Field Service.

For each of these products, use this guide to install QXtend Inbound, then turn to the appropriate section in the guides for these products to complete the specific configuration requirements for QXtend Inbound.

Implementation Overview

This guide provides complete sequential instructions to perform a new QAD QXtend Inbound installation. Implementing a new QXtend Inbound environment entails the following stages:

- Setting up your environment and recording environment information for QXtend Inbound
- Installing the QXI software and related components
- Configuring components in the QXI system to work together

Deployment Options

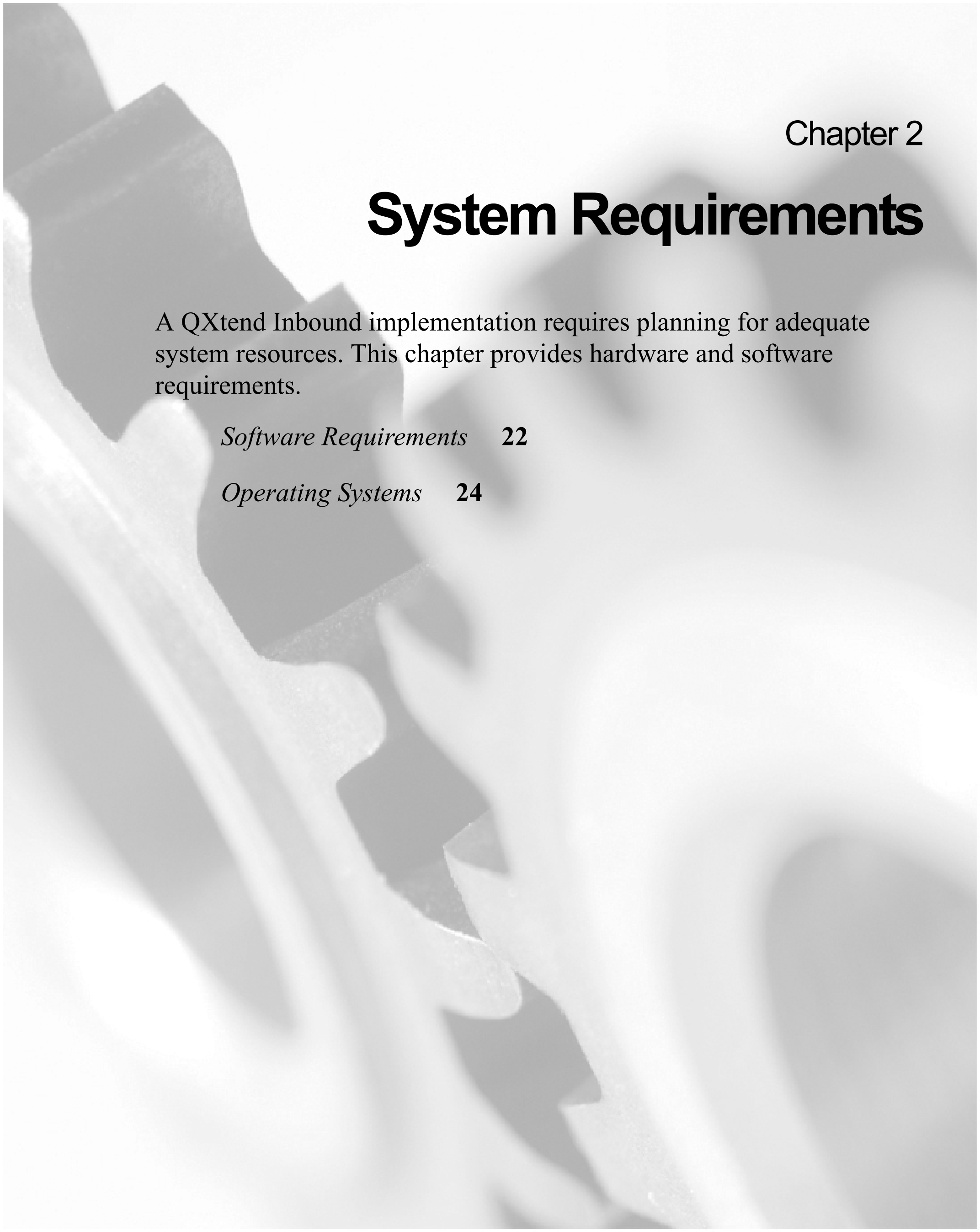
The various components of MFG/PRO and QXtend Inbound can be deployed in several configurations. You can install all of the components on a single adequately sized server in a unified deployment or on multiple servers in a distributed deployment. Both deployments offer advantages for different MFG/PRO environments.

Because deployment options vary widely, these installation instructions assume a unified deployment method is being implemented.

Important Only specially trained and qualified persons with sufficient training and background should install QXtend Inbound in a distributed environment. Contact QAD Global Implementation Services for assistance or QAD Global Learning Services for training information.

Conversions

If you are upgrading from an earlier version of QXtend Inbound, you first back up your custom schemas and events files and the configuration files. You then run through the standard installation process for the new version and restore your custom files to the new implementation afterwards.



Chapter 2

System Requirements

A QXtend Inbound implementation requires planning for adequate system resources. This chapter provides hardware and software requirements.

Software Requirements

This chapter lists the prerequisite requirements to install, configure, and use QXtend Inbound.

QXtend Inbound Baseline Components

QXtend Inbound requires prerequisite components, and includes optional installations on the product media for other required products. A complete QXI installation requires:

- Internet access (Internet Explorer 6)
- Progress 9.1E or Open Edge 10 components including:
  - If your MFG/PRO installation is for a non-US English language, make sure you also install US English from the Progress media. The US English version of PROMSGS is needed for the telnet connection scripts.
  - Latest Progress version-specific patches
- MFG/PRO
  - eB, SP4+
  - eB2, all versions
  - eB2.1, all versions
  - QAD Desktop

See "PROMSGS" on page 159.

For more information, see the MFG/PRO installation guide for your system.

Third-Party Components

- Additional Progress Components

  In addition to the Progress software required for MFG/PRO, QXtend Inbound requires the Progress AppServer, NameServer, and AdminServer if you use the code APIs.
- Tomcat 5 +

  This is available from:

  http://jakarta.apache.org/tomcat/

  You can also use IBM's Websphere as the Web server.
- Java 1.5.0 JDK (for Tomcat 5)

For Linux, Sun, and Windows versions:

http://java.sun.com

For Hewlett-Packard systems:

http://www.hp.com/products1/unix/java/index.html

For AIX systems:

http://www6.software.ibm.com/dl/dka/dka-p

- Telnet server

  QXtend Inbound uses a telnet server to run a pool of telnet sessions for communicating between QXI and MFG/PRO. On UNIX machines, you can use the default telnet service provided with the operating system.

  For Windows systems use:

  Georgia Softworks Telnet Server 6.50 +

  This is available from:

  http://www.georgiasoftworks.com

Supporting Technologies

QXtend Inbound incorporates various Web-based technologies to support various features. These technologies are included with the product and are transparent to the user. They are listed here to give credit to the open-source projects that created them.

- Struts is an open source framework for building Web applications, part of the Jakarta Project, sponsored by the Apache Software Foundation.

  http://jakarta.apache.org/struts/index.html

- Apache AXIS is an implementation of the SOAP (Simple Object Access Protocol) submission to W3C.

  http://ws.apache.org/axis

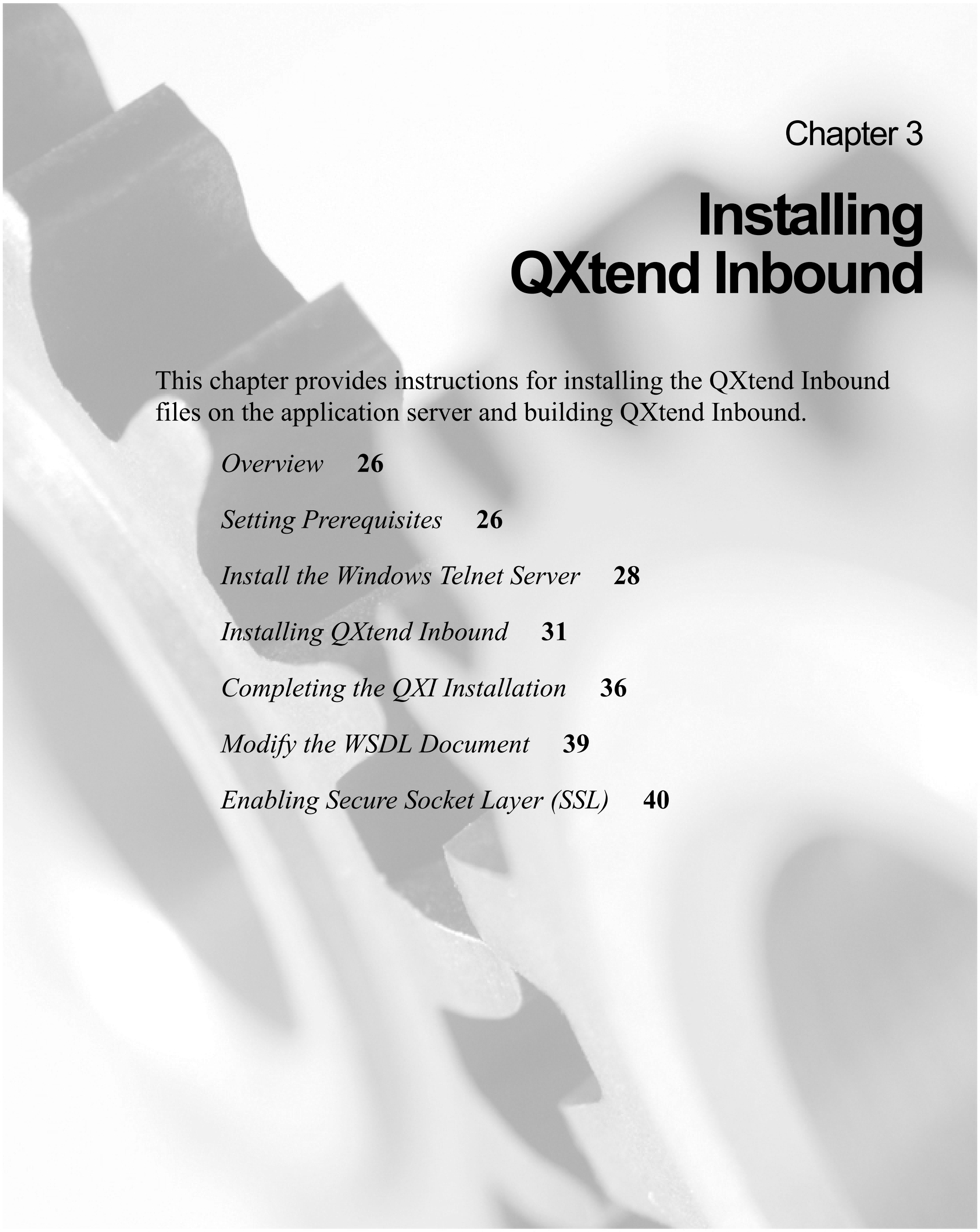
- All QDoc requests and responses are logged using Log4j from Apache, a reliable, fast and flexible logging framework for Java.

  http://jakarta.apache.org/log4j

Operating Systems

QXtend Inbound supports the following platforms:

- Linux
- HP-UX
- Sun Solaris (SPARC)
- Compaq UNIX (Tru64)
- IBM AIX
- Windows



Chapter 3

Installing QXtend Inbound

This chapter provides instructions for installing the QXtend Inbound files on the application server and building QXtend Inbound.

Overview

The QXtend Inbound installation requires the following steps:

- Complete prerequisite setup steps.
- Install the Windows Telnet Server (Windows only).
- Run the install script to prepare the installation environment.
- Install QXtend Progress code and compile.
- Install QXtend Inbound Java files.
- On MFG/PRO eB, load administrative data.
- Compile QXtend Inbound.
- Test the installation and load license codes.
- Optionally, set up QXI to be called as a Web service.
- Optionally, set up QXI for use with Secure Sockets Layer (SSL).

The following installation instructions apply to Windows and UNIX environments.

Setting Prerequisites

Set Environment Variables

In order for the QXI installation scripts to run successfully, the following environment variables need to be set up correctly:

- TOMCAT_HOME
- JAVA_HOME
- CATALINA_HOME

In UNIX environments, add the following lines to the Tomcat user `.profile` to ensure the correct path with each log-in. This may be the `mfg` user or `root`:

```
export TOMCAT_HOME=/pathtoTomcat
export JAVA_HOME=/pathtoJDK
export CATALINA_HOME=/pathtoTomcat
```

To add the environment variables in Windows:

1. Open System in Control Panel.
2. On the Advanced tab, choose Environment Variables.
3. Choose New to add a new variable name and enter the variable value. For example, enter:
   - TOMCAT_HOME variable name; /*pathtoTomcat* variable value
   - JAVA_HOME variable name; /*pathtoJDK* variable value
   - CATALINA_HOME variable name; /*pathtoTomcat* variable value
4. Update the Path in the System Variables to include the two new environment variable values:

   a. Highlight the Path system variable and choose Edit.

   b. Append /*pathtoTomcat*;/*pathtoJDK* to the end of the path.

   c. Choose OK**.**

Set Up Tomcat Security

QXtend Inbound security is implemented through the Tomcat application server. To support the use of QXI with Tomcat, you need to create an administrative log-in and then assign users to it.

Create Administrative Role

1. Create a role, qadadmin, in `tomcat-users.xml` in the *TOMCAT_HOME*/`conf` directory.
2. Define an administrative ID and password for the role in the file:

```
<user username="admin" password="mfgpro" roles=
"qadadmin,admin,manager"/>
```

   If the admin user is already defined, delete or comment out the line.

Install the Windows Telnet Server

For telnet connections on Windows servers, install the Georgia Softworks Telnet Server (GSWTS). The latest version can be downloaded from the Georgia Softworks Web site.

Install the Telnet Server

1 If you are using the QAD-provided version, copy the release-specific `.zip` file from the `/tools` directory on the QAD UI CD to a local drive.

2 Unzip the `.zip` file to a work directory.

3 In the work directory, double-click `setup.exe` to begin installation.

4 At the Welcome window, click Continue.

5 Select Full Install and click Continue.

6 Enter the GSWTS installation directory and choose Continue. Use the default `\gs_uts` installation directory. You can place this directory on any drive that can be accessed through the network.

7 When GSWTS is installed, the Setup Succeeded window displays and GSWTS starts automatically.

Register the Georgia SoftWorks Software

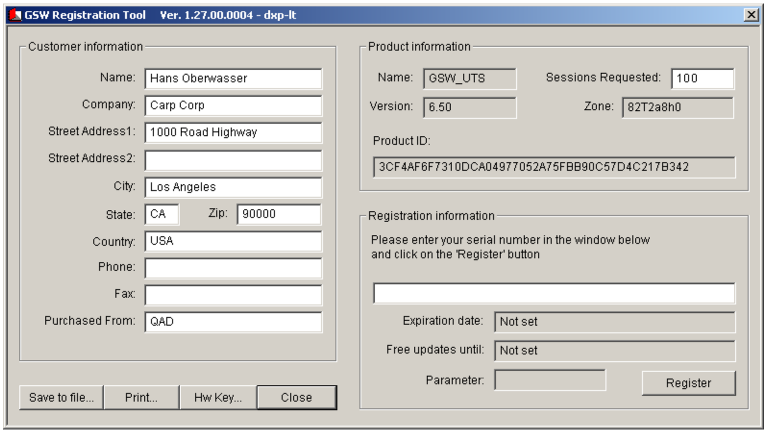
To register GSWTS, you provide a product ID to Georgia SoftWorks so that a serial number can be generated for your product. The serial number identifies server hardware and software components. If these components change or are upgraded, contact Georgia SoftWorks about generating a new product ID and serial number.

Important If you need to reinstall or are planning to move your installation to a different platform, or if you are a sales agent or a distributor, include that information on the registration.

1 Select the Registration icon from the GSWTS program group in the Start menu.

2 In the Georgia SoftWorks Product Registration window, enter your customer information. The information that displays in the Product Information section is system-generated.

3 Set Sessions Requested to 100. This is the number QAD automatically supplies with your registration.



GSW Registration Tool Ver. 1.27.00.0004 - dxp-lt

Customer information

Name: Hans Oberwasser
Company: Carp Corp
Street Address1: 1000 Road Highway
Street Address2:
City: Los Angeles
State: CA Zip: 90000
Country: USA
Phone:
Fax:
Purchased From: QAD

Product information

Name: GSW_UTS Sessions Requested: 100
Version: 6.50 Zone: 82T2a8h0
Product ID:
3CF4AF6F7310DCA04977052A75FBB90C57D4C217B342

Registration information

Please enter your serial number in the window below
and click on the 'Register' button

Expiration date: Not set
Free updates until: Not set
Parameter:

Fig. 3.1
Georgia Softworks Registration

4 Choose Save to File to save this information, or choose Print. Then, follow the appropriate step to supply the product ID to Georgia SoftWorks:

a E-mail the saved registration form file to Georgia SoftWorks at:

registration@georgiasoftworks.com

When your form is received, a serial number is generated for your product and is returned to you by e-mail.

b FAX the printed registration form to Georgia SoftWorks at 706-265-1020. When your form is received, a serial number is generated for your product and is returned to you by FAX.

5 When you receive your serial number, return to the Georgia SoftWorks Product Registration window and enter it in the appropriate field in the registration form. Click Register.

Configuring Georgia Softworks

The Georgia Softworks start-up batch file, `k_start.bat`, must be created manually. In addition, if the QAD Desktop and QXtend Inbound reside on the same host and use the same instance of Georgia Softworks, two `k_start.bat` files must exist, each identified by a separate user directory under the `gs_uts\scripts` directory.

For this second case, there are multiple methods for setting this up. See the Georgia Softworks documentation that downloads with the product in the `\docs` directory for details.

The batch file to launch the Georgia telnet server must be created by hand. Open a text editor and enter the following:

```
@echo off
set gwtn_term=1
set gwtn_graphics=3
set gwtn_color=0
set LRA_TERMINATION=SIG-CTRL-C,SLEEP5,SIG-CTRL-C,SLEEP5,F4,
SLEEP5,e,x,i,t,ENTER,SLEEP5,e,x,i,t,ENTER,SLEEP5,e,x,i,t,ENTER
C:
set prompt=$G
```

Make sure the drive designation, shown in the file sample above as `C:`, is the drive where you have installed Georgia Softworks.

Save the file as `k_start.bat` in the Georgia Softworks scripts directory, *GSWTSHome*`\scripts`. For example, `c:\GS_UTS\scripts`.

Note `gwtn_graphics=3` is set to 6 for QAD Desktop installations.

Installing QXtend Inbound

Install QXtend Inbound Files

For UNIX and Linux installs, complete the install as user `mfg`. This user should have been created as part of a standard MFG/PRO install.

Note On Linux or UNIX systems, the install user must have permission to write to the Tomcat directory in order to create the QXtend environment there.

1. Mount the QXtend Inbound CD-ROM.

2. Navigate to the `install` directory on the CD-ROM.

3. Launch the installation script in that directory. For UNIX, enter:

   ```
   ./install.ksh
   ```

   For Windows, run `install.exe`.

   A welcome screen displays.

4. Accept the default or enter the installation log files location. Later installs look in this location for installation information. If you enter a different log file location, make note of it for later installations.

   ```
   Please enter a directory to write log files
   Default is c:\instlog
   ->c:\instlog
   ```

   The log directory is used to record information about this installation. If you choose not to use the default location, remember the location that you choose so you can use the same location during installation of subsequent media.

   If the directory does not exist, it is created. Confirm the creation by entering `y` and pressing Enter.

5. Review and accept the MFG/PRO software license agreement. Press `q` to jump to the end of the agreement.

   ```
   Do you accept all the terms of the preceding License
   Agreement?
   If you choose no, the install will stop.

   Default is n
   ->y
   ```

6 The main menu displays.

```
*** Main Menu ***
Please choose one of the following:
    1: Install MFG/PRO - Progress QXtend Inbound Files
    2: Install qxtendserver.war
    3: Exit
<1-3>? 1
```

Install QXtend Inbound MFG/PRO Code

Complete this section to install QXI-specific code for MFG/PRO that is included on the QXI media.

1 From the main menu in the installer, choose 1 to begin installing the MFG/PRO Progress files.

2 Enter the MFG/PRO character client installation directory.

In a typical MFG/PRO system, you install the character-client files in the `mfgsvr` directory.

```
Please specify the MFG/PRO Character Client directory.
No default value
        ->C:\mfgsvr
```

3 Enter and confirm the directory for installing QXtend Inbound Progress code. This directory name defaults from the QXI configuration name being installed, by default `qxtendserver`.

```
Please specify the MFG/PRO QXtend Inbound code directory.
The QXtend Inbound Progress code files will be extracted here.
Default is 'C:\mfgsvr\qxtendserver’
->
```

If the directory does not exist, it is created. Confirm the creation by entering `y` and pressing Enter.

4 You are asked to specify MFG/PRO service pack level. The following example is for eB2.1. For an SP3 or SP4 installation, you would choose Other.

```
Please selected installed MFG/PRO Service Pack version.
Please choose one of the following:
1. Other
2. SP0
3. SP1
4. SP2
5. Cancel
```

5 Enter the service pack selection and choose enter. The extraction process starts and is logged to the screen. Files are copied and permissions are set correctly. You are prompted to press Enter as each step completes.

When the process completes, review the log for errors.

This QXtend Inbound directory under the *MFGPROInstallDir* is referred to as *QXIInstallDir* in remaining sections of this document.

6 Review the qxtendserverinstall.log in the log directory to verify the installation. This directory was specified in step 4 on page 31.

Install Tomcat Components

1 On the main menu, choose 2 to install the QXtend Inbound webapp.

Note If the JAVA_HOME variable is not set, you are prompted to set it before continuing. If it is set, you are prompted to confirm.

2 You are prompted to enter and verify the directory where Tomcat is installed.

```
Please enter the Tomcat installation directory.
No default value
        ->c:\Tomcat 5.5
```

3 Enter the port number Tomcat is currently configured to use. This information is needed to stop Tomcat in the next step.

```
Please enter the Tomcat port number.
Default is '8080'
        ->
```

4 The script attempts to stop the Tomcat service before proceeding. If Tomcat is not running, a multiple-line java.net.ConnectException error is raised. The error can be ignored; Tomcat is not running.

5 The installation files are extracted. The setenv.sh and setenv.bat scripts are also updated with environment information. You are prompted to press Enter as each step completes.

The following tasks are completed by the script:

- a The installation script checks for `setenv.bat` and runs it; if it is not found, the installation continues.
- b If this is an update, the `connectionManagerConfig.xml` file, the global `implementationQdocs.xml` file, the `/receivers` directory, the Queue Manager, custom schemas, and custom events are backed up by the script so you do not lose any custom files.

Note The Queue Manager is not automatically backed up if you have changed the default installation paths.

- c The `qxtendserver.war` file is copied to:

  TOMCAT_HOME/webapps

- d The `qxtendserver.war` file is extracted, creating the QXI directory structure under *TOMCAT_HOME*/webapps.
- e If anything was backed up, these files and directories are restored.

See Figure 1.3, "QXtend Inbound Directory Structure," on page 17.

6 Choose 3 on the main menu to exit the install script.

7 Review the `qxtendserverinstall.log` in the log directory to verify the installation. This directory was specified in step 4 on page 31.

8 Restart Tomcat.

Load QXtend Inbound Admin Data (eB only)

If you are installing QXI with MFG/PRO eB, additional data must be loaded. Skip this step for eB2 and later MFG/PRO versions.

1 Launch MFG/UTIL for the MFG/PRO database server.

2 Open Database|Service Pack Process|Process Service Pack Data.

3 The Load Service Pack Data dialog displays.

- a In the Data Directory field, enter the path to the `qxdata` directory installed on the file server; this is typically:

```
/mfgsvr/qxtendserver/qxdata
```

b A list of the databases configured on your system is displayed below this. Select the administration databases; for example, `admprod`.

Note The language selection can be ignored; QXI is currently an English-only product.

4 The Connect screen for the admin database displays. Choose OK.

5 Processing information displays. When the procedure is complete for all selected databases, choose OK again. The Load Service Pack Data dialog closes.

Compile QXtend Progress Code

For UNIX and Linux systems, run the following compile as user `mfg`.

1 In a character session of MFG/UTIL, open Programs|Compile Procedures.

2 Leave the radio buttons with their default settings. In the remaining fields, enter the following data:

- Compile List File

  ```
  /QXIInstallDir/qxtendcompile.wrk
  ```

  For example:

  ```
  /mfgsvr/qxtendserver/qxtendcompile.wrk
  ```

- Compile Propath (ensure the Qxtend path is entered first)

  ```
  /QXIInstallDir/qxxrc,MFGPROInstallDir/xrc
  ```

  For example:

  ```
  /mfgsvr/qxtendserver/qxxrc,/mfgsvr/xrc
  ```

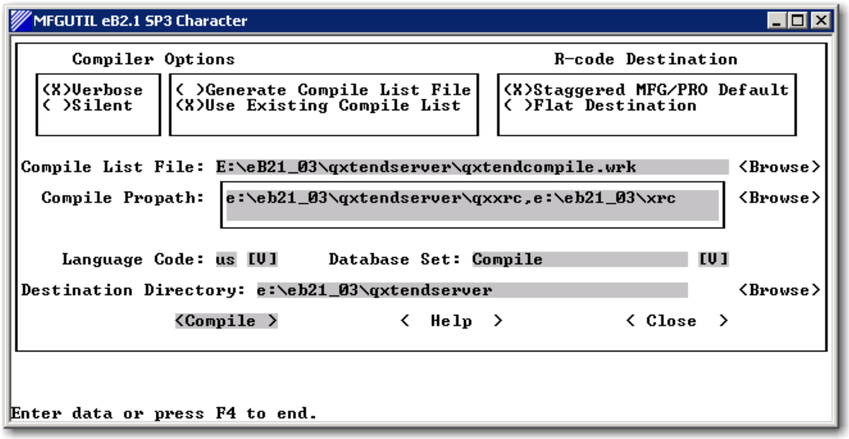
- Destination Directory

  ```
  /QXIInstallDir
  ```

  For example:

  ```
  /mfgsvr/qxtendserver
  ```

Fig. 3.2 Compiling in MFG/UTIL



```
MFGUTIL eB2.1 SP3 Character

      Compiler Options                          R-code Destination
  (X)Verbose  ( )Generate Compile List File  (X)Staggered MFG/PRO Default
  ( )Silent   (X)Use Existing Compile List   ( )Flat Destination

Compile List File: E:\eB21_03\qxtendserver\qxtendcompile.wrk       <Browse>
  Compile Propath:  e:\eb21_03\qxtendserver\qxxrc,e:\eb21_03\xrc    <Browse>

    Language Code: us [V]     Database Set: Compile            [V]
Destination Directory: e:\eb21_03\qxtendserver                      <Browse>
               <Compile >            <  Help  >           < Close  >

Enter data or press F4 to end.
```

3 Choose Compile.

4 Verify all information is correct in the summary screen and choose Continue.

Completing the QXI Installation

The final steps in the QXtend Inbound installation are to verify that you can log in to the application and to load license codes.

Log In

To test that the Java components of QXtend Inbound are correctly installed:

1 Start Tomcat if it is not already running.

2 Launch a browser on your client machine.

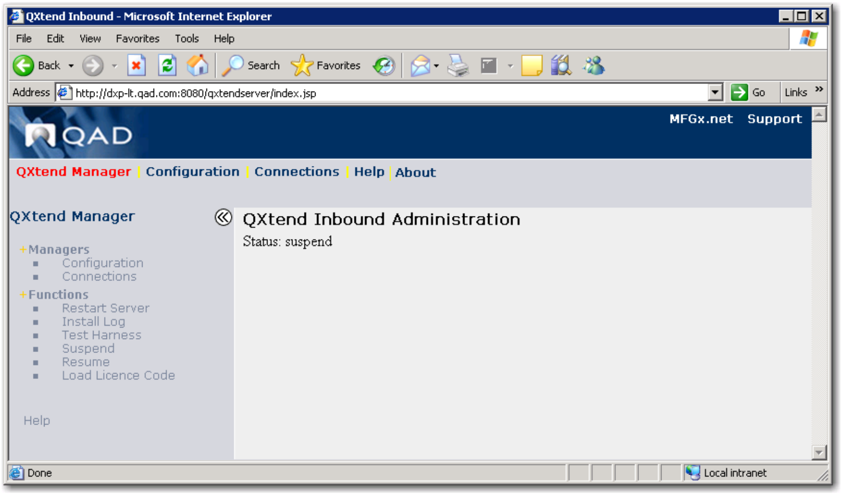
3 Navigate to the following URL:

```
http://<qxtendhost>:8080/qxtendserver/index.jsp
```

The login screen displays. Enter the admin user and password defined previously. Choose OK.

See “Setting Prerequisites” on page 26.

You should see screen illustrated in Figure 3.3.


QXtend Inbound - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://dxp-lt.qad.com:8080/qxtendserver/index.jsp
Go Links
QAD
MFGx.net Support
QXtend Manager | Configuration | Connections | Help | About
QXtend Manager
Managers
Configuration
Connections
Functions
Restart Server
Install Log
Test Harness
Suspend
Resume
Load Licence Code
Help
Done
Local intranet

QXtend Inbound Administration

Status: suspend

Fig. 3.3
QXI Main Screen

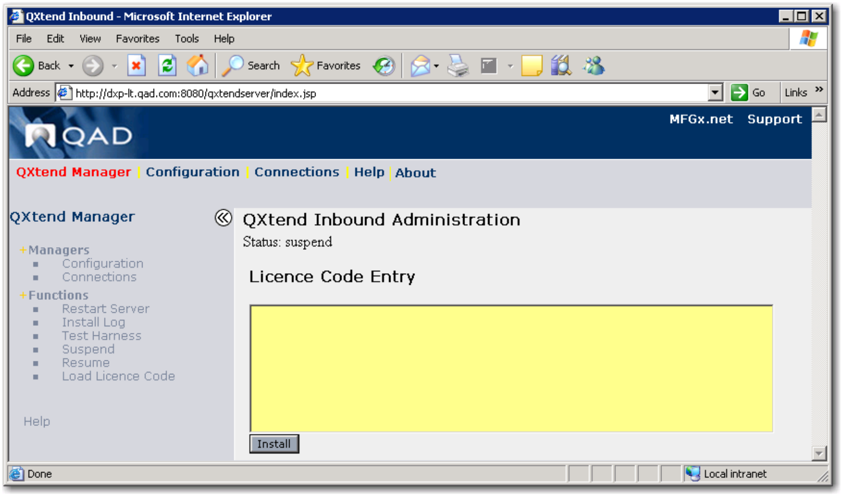
Load License Codes

QXtend Inbound is limited to licensed requestor types such as MFG/PRO and JIT/S. The first time you access the application, the QXtend Manager link at the top is red, rather than blue, indicating that the application has not been enabled by loading license codes.

To add a new requestor license, choose Load License Code in the QXtend Inbound Administration page.

Note If you did not receive license codes, contact QAD Support.

Fig. 3.4
License Code Entry Form



Each receiver is identified in the incoming SOAP envelope of a request document. QXI validates each receiver against the licensed receivers. Licenses may contain expiration dates.

After you enter a new license, click Install. If the license is successful, the QXtend Inbound administration status changes from Suspend to Active.

Setting Up QDoc Requestors Instead of User IDs

In addition to licensing each QDoc requestor, you can choose to validate QDoc requestors either by validating them using the user ID and password in the Connection Manager, or by sending and validating the requestor user ID and password in the QDoc itself.

Validate requestors through the Connection Manager by setting the `useQDocRequestor` value to `false` (the default setting) in the `qxtendconfig.xml` file.

Set the value to `true` to check the `<requestor>` user ID and password entries in an incoming QDoc SOAP envelope. If set to `true`, the user ID and password must be present in the QDoc SOAP envelope, or an error will occur.

Important If this instance of QXtend is intended for use with QAD Production Scheduler, leave the setting as `false`. That application does not create requestor nodes for validation.

1 Open *TOMCAT_HOME*`/webapps/qxtendserver/WEB-INF/conf/qxtendconfig.xml.`

2 Locate the `useQDocRequestor` attribute within the `<general-config>` node.

3 Change the value from "`false`" to "`true`".

4 Save and close the file.

Modify the WSDL Document

In order to directly call QXtend Inbound as a Web service, a client component must be developed that is capable of calling QXtend using the SOAP and HTTP protocols. To generate the code, most Web service tools require a special XML document written in the Web Services Description Language (WSDL).

See "WSDL Document for Web Service Clients" on page 15.

QXtend Inbound provides a single WSDL document that you can modify manually to designate the network address (URL) in the QXtend Inbound installation where the Web service will run. This file is:

```
TOMCAT_HOME/webapps/qxtendserver/WEB-INF/
myQdocWebService.wsdl
```

This WSDL document must be modified manually. To modify it, do the following:

1 Make a backup copy of `myQdocWebService.wsdl`; for example, `myQdocWebService.orig`.

2 Open the original `myQdocWebService.wsdl` in a text editor.

3 There are two occurrences of `myQdocWebServiceAddress` at the bottom of the file. Change these to *`<hostname>:<tomcat_port>`*, where *`<hostname>`* is the host name or IP address of the computer where QXtend Inbound is installed, and *`<tomcat_port>`* is the port number at which Tomcat is listening for requests.

```
<wsdl:service name="myQdocWebService">
    <wsdl:port binding="intf:QdocWebServiceSoapBinding" name=
    "webhost.corp.com:8080">
        <wsdlsoap:address location="http://
        webhost.corp.com:8080/qxtendserver/services/
        QdocWebService"/>
    </wsdl:port>
</wsdl:service>
```

Note The file assumes that the path name of the Web service component within *TomcatInstallDir*/webapps is:

/qxtendserver/services/QdocWebService

4 Save the changes to myQdocWebService.wsdl.

The QXtend Inbound Web service consists of a single operation called processQdocMessage that accepts two parameters: a SOAPEnvelope object representing the QDoc request, and a SOAPEnvelope object representing the QDoc response.

Unlike many other Web services, each QDoc is not handled by a separate operation within the Web service. Also, the QXtend Inbound Web service is defined as a document-style rather than RPC-style (Remote Procedure Call-style) Web service.

Most of the QDocs are relatively complex XML documents with many elements, levels, and parent-child relationships among elements, as required to represent the underlying MFG/PRO application data and business logic. They are not queries with a small number of parameters cast from simple data types, as is the case with most RPC-style Web services.

Enabling Secure Socket Layer (SSL)

QXtend supports Secure Socket Layer (SSL) between the client and the server to encrypt all HTTP messages between the client and server. This requirement can be satisfied by correctly configuring Tomcat to provide an SSL connection to the QXtend server Webapp.

To complete the configuration, you must:

- Modify server.xml.
- Modify catalina.bat or catalina.sh.
- Create trusted security certificates in Java.

Modify server.xml

The first step is to define an SSL HTTP/1.1 Connector on port 8443 in the `TOMCAT_HOME\conf\server.xml` file. The first section of the file content below appears in the default `server.xml`. Add the Connector definition as shown following the end of the comment marker (-->).

```
<!-- Define a SSL HTTP/1.1 Connector on port 8443 -->
<!--
<Connector port="8443" maxHttpHeaderSize="8192"
    maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
    enableLookups="false" disableUploadTimeout="true"
    acceptCount="100" scheme="https" secure="true"
    clientAuth="false" sslProtocol="TLS" />
-->

<Connector
    className="org.apache.coyote.tomcat4.CoyoteConnector"
    port="8443"
    scheme="https"
    secure="true"
    useURIValidationHack="false"
    disableUploadTimeout="true">
</Connector>
```

Refer to the Tomcat documentation for further information.

Modify catalina.bat or catalina.sh

For Tomcat versions prior to 5.5, do the following.

In order for the QXtend servlets to work with the HTTPS connector, properties must be set in `TOMCAT_HOME/bin/catalina.bat` for Windows and `TOMCAT_HOME\bin\catalina.sh` for UNIX. The QXtend servlets properties are:

```
-Djavax.net.ssl.trustStore=<Location of the keystore>
-Djavax.net.ssl.trustStorePassword=<keystore password>
-Djava.protocol.handler.pkgs=com.sun.net.ssl.internal.www.protocol
```

Enter these lines into the file after each instance of RUNJDB or RUNJAVA. One entry is shown below as an example; however, there are typically seven or eight locations where the additions must be made.

```
exec "$_RUNJDB" $JAVA_OPTS $CATALINA_OPTS \
    -Djavax.net.ssl.trustStore=<Location of the keystore> \
    -Djavax.net.ssl.trustStorePassword=<keystore password> \
    -Djava.protocol.handler.pkgs=
    \com.sun.net.ssl.internal.www.protocol \
    -Djava.endorsed.dirs="$JAVA_ENDORSED_DIRS" -classpath
    "$CLASSPATH" \
    -sourcepath "$CATALINA_HOME"/../../jakarta-tomcat-4.0/
    catalina/src/share \
    -Djava.security.manager \
    -Djava.security.policy=="$CATALINA_BASE"/conf/
    catalina.policy \
    -Dcatalina.base="$CATALINA_BASE" \
    -Dcatalina.home="$CATALINA_HOME" \
    -Djava.io.tmpdir="$CATALINA_TMPDIR" \
    org.apache.catalina.startup.Bootstrap "$@" start
else
exec "$_RUNJDB" $JAVA_OPTS $CATALINA_OPTS \
    -Djavax.net.ssl.trustStore=<The location of the keystore> \
    ...
```

Create Trusted Security Certificates

The machine on which Tomcat is running requires a trusted security certificate. If the HTTPS client cannot find a trusted security certificate on the QXtend server, it may issue a warning to that effect or may refuse to work with a HTTPS connection to the server. This behavior can cause a fault on the client side.

A self-signed server certificate must be generated with the Java keytool. The keytool program prompts you for several values.

1. In a command window, navigate to a temporary directory such as c:\temp and run the following command:

   ```
   %JAVA_HOME%\bin\keytool -genkey -alias tomcat -keyalg RSA
   -keystore c:\temp\.keystore
   ```

2. You are prompted to enter the keystore password. The default SSL password for Tomcat is `changeit`. If you change this, you need to place the new password in `server.xml` and `catalina.bat`.

3. You are prompted to enter your first and last name. Enter the name of the host where the QXtend Inbound webapp is installed; for example, `corp_pc01`. It must be the same host specified in `qxtendconfig.xml` for MessageReceiverServlet. This and all following prompts display with the following format:

   ```
   [Unknown]: <Enter data value here.>
   ```

4. You are then asked for the following information for which you can enter the appropriate values for your implementation:
   - The name of your organizational unit
   - The name of your organization

- The name of your city or locality
- The name of your state or province
- The two-letter country code for this unit

5 You are then asked to confirm:

```
Is CN=corp_pc01, OU=QAD, O=QAD, L=Summerland, ST=
California, C=US correct?
```

```
[no]:  y
```

The common name (CN) must be the correct QXI webapp host or the Web service call raises a validation error: HTTPS hostname wrong.

6 You are then prompted to enter the key password for <tomcat>. Press Return if the key password for Tomcat is the same as the keystore password. Do this unless you are experienced with HTTPS and know what you are doing.

See the Java tools documentation for more information on this topic.

Configure the MessageReceiverServlet URL

Under certain conditions, Progress send back messages to QXtend through the MessageReceiverServlet. This requires an HTTP connector because Progress sockets do not support HTTPS.

In the current implementation, the MessageReceiverServlet URL is built from parameters in the systemdetails.info file, which is populated from the URL. This file is located in *TomcatInstallDir*/webapps/qxtendserver/WEB-INF. For the implementation of SSL, this information is added to the qxtendconfig.xml file.

Configuring qxtendconfig.xml

Add a messageServletURL element to the *TOMCAT_HOME*/webapps/qxtendserver/WEB-INF/conf/qxtendconfig.xml file.

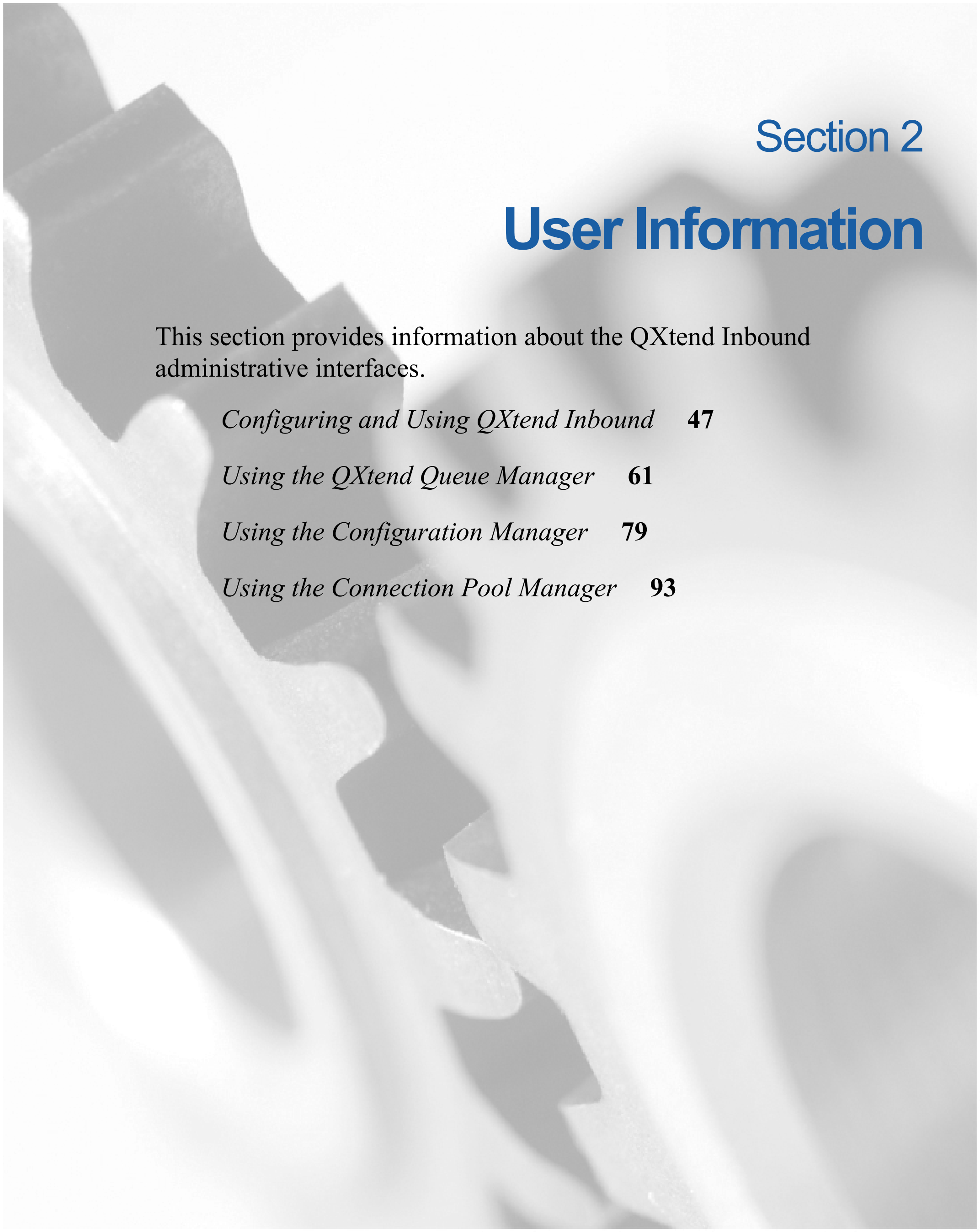
```
<!-- The URL of the message servlet -->
    <messageServletURL label="Message Servlet URL" value="https:/
    /127.0.0.1:8080/qxtendserver/MessageReceiverServlet"/>
```

The 127.0.0.1 element should be configured for the individual machine on which QXI is installed. This should be the machine name or IP address of the host, and not localhost.

The port value, 8080, should be changed to the HTTPS port, 8443.

QXtend can then support SSL and can be reached through a URL something like the following:

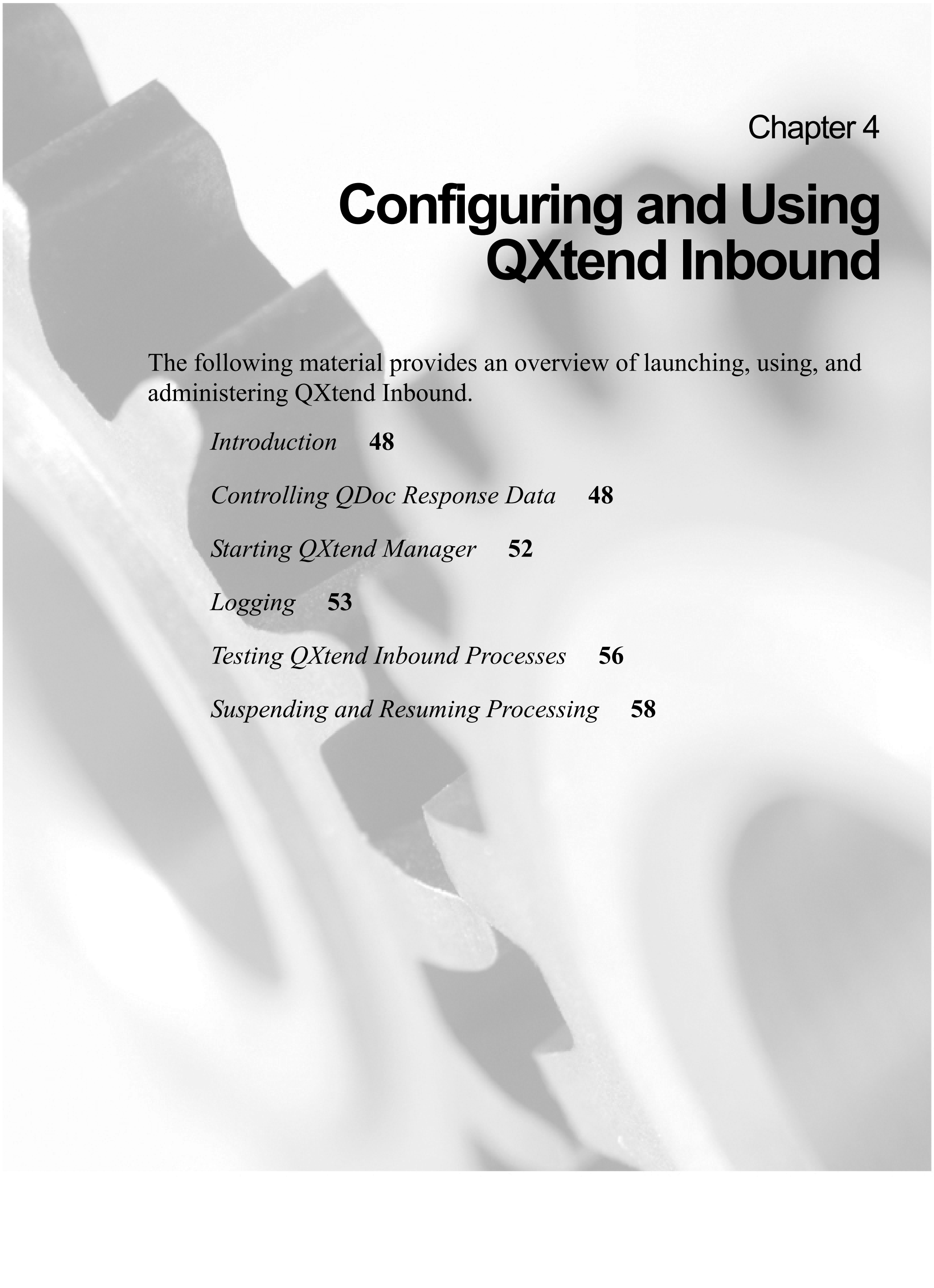
https://*webhost.corp.com*:8443/qxtendserver/html/test/index.html



Section 2

User Information

This section provides information about the QXtend Inbound administrative interfaces.



Chapter 4

Configuring and Using QXtend Inbound

The following material provides an overview of launching, using, and administering QXtend Inbound.

Introduction

QXtend Inbound provides an interface called the QXtend Manager to complete administrative tasks for QXtend Inbound. The QXtend Manager consists of a Configuration Manager, a Connection Pool Manager, and an optional Queue Manager and functions to restart the server, view the install log, and run programs in the QXI test suite.

Use the Configuration Manager (discussed in Chapter 6) to manage your QDoc schemas, to create and modify your MFG/PRO receivers, and to allocate QDoc schemas to the receivers. Use the Connection Pool Manager (discussed in Chapter 7) to create, view, and control the available telnet connections to your MFG/PRO sessions.

➧ See Chapter 5, "Using the QXtend Queue Manager," on page 61.

An additional interface that is not a required component of QXtend Inbound is the Queue Manager. This tool provides a QDoc queuing service, creating request and response directories for each external application that is sending QDoc requests. The request QDocs are picked up from a specified request directory; the response QDocs are returned by the Queue Manager to a specified response directory where the external application picks them up.

Controlling QDoc Response Data

➧ For additional information on specific exceptions included in response QDocs, see Chapter 16.

By default, QDoc response documents contain success and failure information. They can also contain additional warnings and errors, Java trace information, and primary key and other field values from MFG/PRO.

Default Warnings and Errors

Additional warnings and errors are:

- Notifications from pre- and postprocessing programs
- Occurrences of unused fields in MFG/PRO
- Occurrences of unused iterations in MFG/PRO
- Occurrences of overlength field data in MFG/PRO

Pre- and Postprocessing Messages

The content of the pre- and postprocessing messages is up to the programmer writing the programs.

For details on how to make the correct calls to add these messages to the response, see "Warnings and Errors" on page 133.

Messages for Unused Fields

In some cases, fields may exist in a QDoc that are never used during the processing of that QDoc. For example, if a field node exists in the QDoc that is not part of the schema or is misspelled, it is not matched with a field in MFG/PRO and remains unused during the processing of that QDoc. This information is added to the response QDoc as a warning regardless of the `suppressResponseDetail` setting (see "Include MFG/PRO Field Values" on page 50). The response message is:

```
Field was unused, see context for details.
```

Messages for Unused Iterations

In some cases, entire iterations in a QDoc are never used during the processing of that QDoc. For example, a control program disables the display of a frame but an iteration with fields for that frame is included in the QDoc. Since that frame is never visited during the processing of the QDoc, a warning appears in the response QDoc regardless of the `suppressResponseDetail` setting (see "Include MFG/PRO Field Values" on page 50). The response message is:

```
Iterations were unused, see context for details.
```

For both unused fields and unused iterations, if the QDoc is deleting a record and if additional fields or iterations are in the QDoc, these are noted by messages in the response QDoc also. It is usually the case that only a smaller subset of fields is needed to delete a record, but users may not always remove the unnecessary fields or iterations from the schema files.

Messages for Field Length

Occurrences where the data length of the input data from the QDoc is too long for the input field in MFG/PRO cause either a warning or error message, depending on whether the field is a primary key. If it is a primary key, an error is returned; otherwise, a warning:

```
Data submitted to MFG/PRO was too long for the MFG/PRO
field size.
```

Include Java Trace Information

You can include Java trace information in the response documents in order to debug processes. To do this, edit the `qxtendconfig.xml` file located in `*TOMCAT_HOME*/webapps/qxtendserver/WEB-INF/conf`. In it, locate the node containing:

```
boolean excludeTraces = “true”
```

To include the Java trace information, change this to:

```
boolean excludeTraces = “false”
```

Include MFG/PRO Field Values

By default, the QDoc response document contains the primary key values from each iteration (for example, sales order number and sales order line item values). You can edit the response type schema to eliminate or add field values you want to see.

However, by default, no values are returned in the response document. Turn the option on by editing the specific QDoc. Locate the following attribute:

```
suppressResponseDetail = “true”
```

Change the setting to:

```
suppressResponseDetail = “false”
```

See “Add a Schema to the Master Lists” on page 87 for steps.

To control which field values are returned, create a copy of the response type schema file (`salesOrderResponseType-eB2_1.xsd` for `sosomt.p`) for the specific response document and edit it. Comment out the fields you do not want to see in the response document, and add in any fields you want to add, then save it as a custom schema file. You then assign the response schema file to the master schema lists where it can be assigned to a specific receiver.

Note To add fields not already referenced in the response schema, you can copy them from the request type schema.

To enable the response communication in QXtend Inbound, edit the `qxtendconfig.xml` file:

1 Open *`TOMCAT_HOME`*`/webapps/qxtendserver/WEB-INF/conf/qxtendconfig.xml`.

2 Locate the `messageServletURL` attribute within the `<general-config>` node.

3 Enter the correct host and port values to communicate with the QXI server.

```
<messageServletURL label="Message Servlet URL" value="http:
//<host>:<port>/qxtendserver/MessageReceiverServlet"/>
```

4 Save and close the file.

Performance Considerations

Adding fields to the response QDoc schemas may have an impact on performance. In addition, performance may be impacted by the fields contained in the schema file by default.

The fields in the QAD-generated response schemas contain all key data for that API, that is, all primary fields for all iterations. These fields are all typically updated during a QXI update. However, including additional non-primary key fields requires QXI to navigate to these fields. This can impact performance if the field is not always updated, or when it requires additional navigation steps to reach the field in MFG/PRO.

Example The sales order response schema is modified to include the Remarks field. However, the request QDoc does not update the Remarks frame. However, QXI still navigates through all necessary frames until the Remarks field is found and the data returned.

This may also be an issue in some standard schemas when a particular iteration is not being updated but the data values are being returned because the response schema requires it.

Starting QXtend Manager

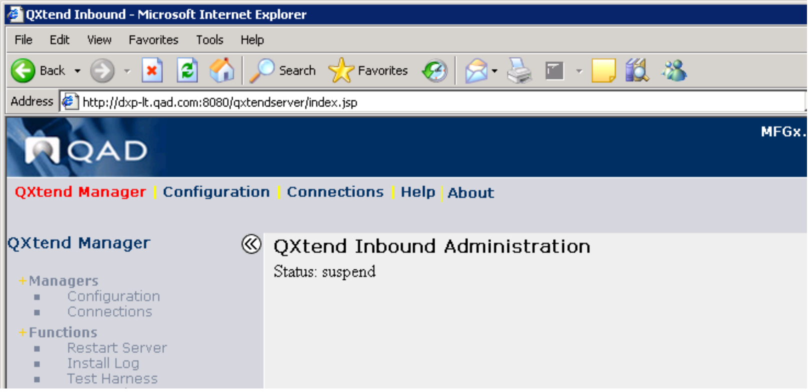
Prior to starting the QXtend Manager, start Tomcat and your target MFG/PRO sessions. Then open a browser and enter the following URL:

```
http://<hostname>:<tomcat_port>/qxtendserver/
```

Replace *<hostname>* with the machine name where Tomcat is installed. Replace *<tomcat_port>* with the Tomcat port number.

The QXtend Manager opens.

Fig. 4.1
QXtend Manager


QXtend Inbound - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://dxp-lt.qad.com:8080/qxtendserver/index.jsp
QAD
MFGx.
QXtend Manager | Configuration | Connections | Help | About
QXtend Manager
Managers
Configuration
Connections
Functions
Restart Server
Install Log
Test Harness
QXtend Inbound Administration

Status: suspend

Access the Queue Manager, Configuration Manager, or Connection Pool Manager either in the menu bar or under Managers. You can also refresh the QXtend Manager interface from the menu bar.

The QXtend Manager Functions are:

> *Restart Server.* This shuts down QXtend Inbound and restarts it, using the Tomcat restart link.
>
> *Install Log.* This displays `qdocInstall.log`. The log is updated each time the QXI Web application is started, stopped, or reloaded. The log is stored in:
>
> *TOMCAT_HOME*/webapps/qxtendserver/WEB-INF/logs

Test Harness. The test harness provides an interface you can use to run through the entire request-response process using sample QDocs.

Suspend. This shuts down the QXtend services to allow an administrative update, such as configuration changes.

Resume. This restores QXI services following administrative changes.

Load License Code. This allows you to enter license codes for valid user types, such as MFG/PRO.

Logging

Logging is facilitated by the Apache Jakarta Project Log4j infrastructure. This means that log statements are in the code; their output can be configured in an external XML file. In QXtend Inbound, this file is called `qxtendlogging.xml` and is located in:

```
TOMCAT_HOME/webapps/qxtendserver/WEB-INF/conf
```

All log files are created and stored by default in:

```
TOMCAT_HOME/webapps/qxtendserver/WEB-INF/logs
```

Unless otherwise described, do not change the details for these logs, except where they are stored or the reporting levels. It may be useful to move these files to backup storage, or to delete them, once the records are no longer required.

Information on how to edit these files is beyond the scope of this document; further information can be found on the Log4j Web site:

http://jakarta.apache.org/log4j/docs/index.html

Log Report Levels

Reporting levels for the QXI logs are:

- Error – error messages only
- Warning – error and warning messages only
- Info – error, warning, and info messages only
- Debug – all messages

These can be set for each of the logs in qxtendlogging.xml.

The logs created in QXtend Inbound are:

- qdocInfo.log
- qdocRequests.log
- qdocResponses.log
- queue.log
- transformationEngine.log
- connectionPools.log
- qdocInstall.log
- qxtendserver.log
- transformationengineRequests.log
- transformationengineResponses.log
- transformationEngine.debug

The first three logs provide backup data for the QDocs going through the server. The logs are either rolled over daily with the date appended to the previous file; for example:

```
qdocInfo.2005-05-29.log
```

Or they are rolled over based on log size. See the following descriptions for details by log.

qdocInfo.log

The qdocInfo log contains an entry for each QDoc received and the document return status. If the return status is an error, the entire QDoc response is entered in the log. A sample non-error entry:

```
2003-05-30 19:51:26,308 INFO  qdocLogger.info [1054320686308]
[-]API Request Received
```

The message line components are ISO format date (2005-05-30), ISO format time (19:51:26, 308), reporting level (INFO), logger (qdocLogger.info), unique message ID (1054320686308), QDoc ID, and message (API Request Received).

The other two QDoc logs—qdocRequests.log and qdocResponses.log—list the complete QDocs sent to the server and response QDocs respectively. The format is the same as for qdocInfo.log.

These files are rolled over daily.

queue.log

This logs activities in the Queue Manager, listing all files it finds in the requests directory. It also lists file name changes—for example, `req -> req_wrk -> ok/err`. This file rolls over daily.

connectionPools.log

This logs connection pool details such as number of connections, changes in connection status, and so on. This rolls over daily.

qdocInstall.log

This log is updated with details when the QXtend Inbound Webapp is started, stopped, or reloading, listing components and whether they initiated or terminated successfully.

This log will have a maximum of five files (current + four previous), with a sequence number. The file is rolled over based on log size, which is currently set to 500K; for example, `qdocInstall.log.1`.

Once the maximum number of files has been created, QXI reuses the existing files. You do not need to delete these files since they always take up finite storage space.

qxtendserver.log

This log lists all errors that have occurred during the running of QXtend Inbound; it is intended mostly as a developer's log. This log has a maximum of five files (current + four previous), with a sequence number, such as `qxtendserver.log.1`. The file is rolled over based on log size—currently set to 1000K. Once the maximum number of files is created, the log reuses existing files.

transformationEngine.log

For details, see "Configuring the Transformation Engine" on page 65.

This log file contains information about key steps in the transformation, namely creating managers. It contains error information when a problem occurs. When set to debug it shows detailed transformation steps.

transformationengineRequests.log

This log maintains a listing of all requests processed by the transformation engine as well as the document contents.

transformationengineRequests.log

This log maintains a listing of all responses processed by the transformation engine as well as the document contents.

transformationEngine.debug

This log contains all debug messages raised during request processing in the transformation engine. This log requires that developers creating transformation engine style sheets add debug messages to their code that are then output to this file.

Testing QXtend Inbound Processes

The QXtend Inbound provides a tool set for testing QXI processes.

Process Request

This option lets you test the installation and configuration of QXtend Inbound. A valid QDoc wrapped in a SOAP envelope can be placed into the `/requests` directory under one of the receivers set up in QXtend Inbound; for example:

```
TOMCAT_HOME/webapps/qxtendserver/WEB-INF/receivers/
<receiverName>/requests
```

In the Process Request test page, enter the file name and the receiver name where you placed the test request. The request is submitted to QXI for processing and the response QDoc is displayed in a graphical format. View the response XML by choosing the View Details button.

Create Empty QDoc

This option creates a sample QDoc based on an XML schema you specify. The schema must be one that is supported by QXtend Inbound. The schema is parsed and all possible entries from the QDoc schema are created in the sample QDoc. The sample QDoc shows the way to represent all of the possible data structures in the QDoc. The generated QDoc shows the structure and content of the QDoc that needs to be sent to QXtend Inbound. The generated schema also contains a valid SOAP envelope.

Verify QDoc Supported

This option checks the standard and custom deployment descriptor files to see if a specified QDoc, QDoc version, and receiver is supported for a specified schema.

Verify Receiver

This option checks the `qdocReceivers.xml` file to see if the specified receiver has been set up.

Create Merged Schema

This function takes the two component XML schemas—base and type—and merges them into a single XML schema file. This tests the QXtend Inbound process that merges component QDoc schemas into a single object.

Map XML Schema to 4GL API

This function converts a QDoc XML schema to a metaschema temporary (temp) table that simulates the data that would be sent to Progress for the AppServer. The contents of this temp table are used in normal processing to create the dynamic temp tables that are required to process code API calls.

Map Request XML to 4GL API

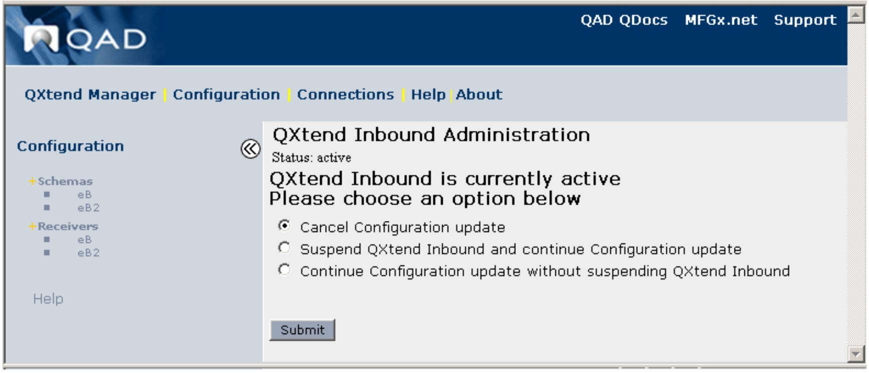
This function parses a request QDoc without a SOAP envelope into a simulation of the data structure that is used to pass the request data to the Progress using the Progress AppServer. The output of this function shows the data that is loaded into dynamic temp tables.

Suspending and Resuming Processing

To avoid processing errors, QAD recommends that you suspend QXI processing prior to modifying schemas, receivers, connections, and other QXI configuration or XML components.

When you initiate an update of one of these QXI components, a warning displays and gives you options for proceeding.

Fig. 4.2
Suspension Options Prior to a Schema Update



QXtend Inbound Administration

Status: active

QXtend Inbound is currently active
Please choose an option below

● Cancel Configuration update
○ Suspend QXtend Inbound and continue Configuration update
○ Continue Configuration update without suspending QXtend Inbound

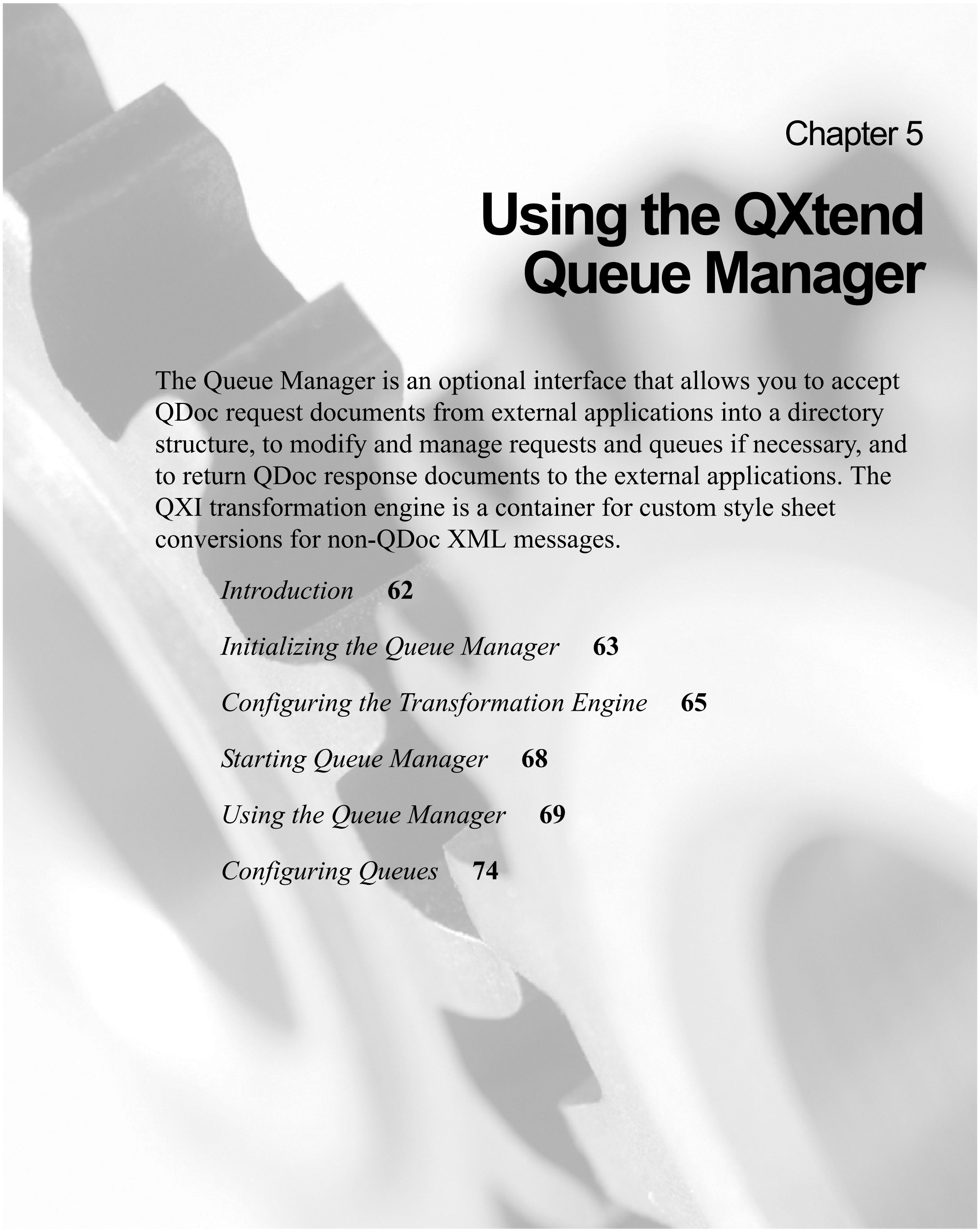
If you choose to cancel the configuration, you return to the previous screen. If you choose to continue without suspending QXI, QDoc requests and responses continue to be processed while you make changes. This course of action may be acceptable if you are modifying new or obsolete configurations not related to any current requests.

If you choose to suspend processing, all current QDocs will complete their processing, and all new QDoc requests will be failed with an error message returned in the SOAP envelope. These requests must be resent after QXI has resumed.

You can also choose Suspend from the QXtend Inbound Administration page.

Resume QXtend

To resume processing, choose Resume from the QXtend Inbound Administration page.



Chapter 5

Using the QXtend Queue Manager

The Queue Manager is an optional interface that allows you to accept QDoc request documents from external applications into a directory structure, to modify and manage requests and queues if necessary, and to return QDoc response documents to the external applications. The QXI transformation engine is a container for custom style sheet conversions for non-QDoc XML messages.

Introduction

You can also use QAD's Q/LinQ product for a more robust queuing and administrative application. See "Using QXtend with Q/LinQ" on page 143.

The Queue Manager is accessed through the QXtend Manager interface, but is not a required or default component of QXtend Inbound. QXtend Inbound accepts API requests directly as HTTP post requests in SOAP-compliant XML format, or can accept the documents from a queuing application such as IBM MQSeries, or from the QXtend Queue Manager.

Queuing applications provide queues for incoming and outgoing documents or messages, ensuring delivery and synchronous responses.

In the case of the Queue Manager, requests arrive from external systems to a specified directory. The requests must be in XML format and be named with a `.req` extension. (The Queue Manager can add a SOAP envelope if necessary.)

For details, see "Configuring the Transformation Engine" on page 65.

Non-standard XML requests can also be received by queues set up to allow them. These are XML documents added to the queue that do not have a `.req` extension. In this case, the document is passed to the transformation engine. The engine applies a custom XSLT style sheet based on the extension of the file and creates a standard QDoc request. It places this request in a standard queue directory where it is processed normally.

The Queue Manager polls the request directory, picks up new requests, renames them with a `.req_wrk` extension, and sends them to QXtend Inbound where they are sent to the defined receiver. A response document container is created at the same time; messages, requested fields, errors, and so on are written to the response during QDoc processing.

Once processing is complete, responses are placed in a response queue for the external application. Responses use the following extensions:

ok. The QDoc processed correctly.

.wrn. The QDoc processed but encountered warnings.

.err. The QDoc request failed.

Once a response document is created in the `\response` directory, the `.req_wrk` request is moved to the response directory and renamed with a `.req` extension again.

If the Queue Manager does not receive a response for some reason, the `.req_wrk` request is moved to the external application `systems_failure` directory.

Multi-Threaded and Single-Threaded Queues

Multi-threading QDoc queues allows a higher throughput and better performance. However, in multi-threaded queues where QDocs have a sequential dependency, it is possible for a dependent QDoc to start processing before the initial QDoc has finished. Therefore, queues are defined with thread sizes to allow you to create multi-threaded queues for QDocs without dependencies; single-threaded queues for QDocs with dependencies.

See “Thread Settings” on page 78.

Initializing the Queue Manager

In order to use the Queue Manager, complete the following steps:

For information on queue errors, see “Queue Exceptions” on page 249.

1 Open `environmentmanager.xml` in:

   TOMCAT_HOME`/webapps/qxtendserver/WEB-INF/conf`

   Instructions for initializing the Queue Manager also appear in the file.

2 Locate the commented section in the `-<managers>` node.

3 Uncomment the `manager class` node following the in-line comments; delete the `<!--` and `-->` symbols.

4 Replace the `param` value with the full path to the top-level Queue Manager directory. You can use any directory structure you choose, on any connected network machine; however, if the full path is not specified in `environmentmanager.xml`, the Queue Manager will not work correctly; for example:

```
<manager class="com.qad.qxtend.queue.directory.DirectoryManager"
param="c:\Tomcat 5.5\webapps\qxtendserver\qxtendQueues"/>
```

5 Save the file.

6 Restart QXtend Inbound in order for the changes to take effect.

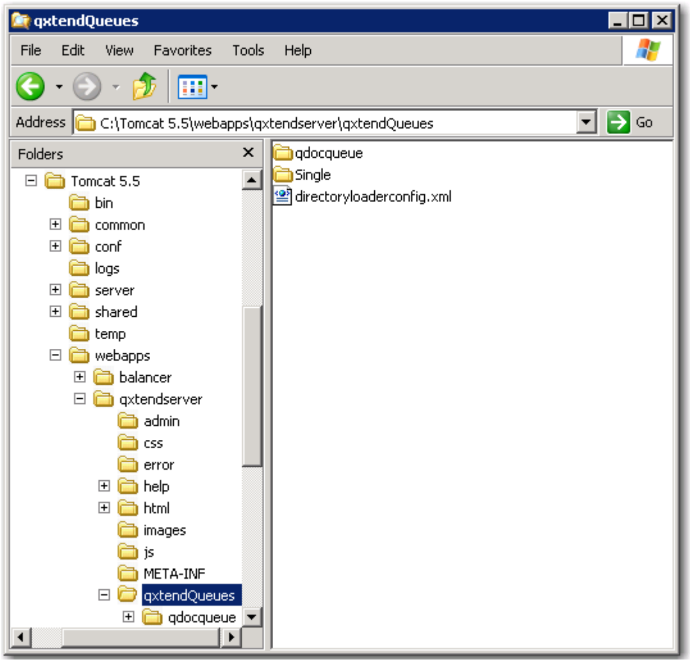
Important It may be necessary to stop and restart Tomcat before the Queue Manager displays.

Queue Manager Directory Structure

The default Queue Manager directory structure is located under:

TOMCAT_HOME/webapps/qxtendserver/qxtendQueues

Fig. 5.1
Queue Manager Sample Directory Structure



Address C:\Tomcat 5.5\webapps\qxtendserver\qxtendQueues

Underneath the base `qxtendQueues` directory are directories for each queue and two configuration files:

> *directoryloaderconfig.xml.* This file contains global configuration details for Queue Manager. These are the location of the Queue Manager log file, `qdocDirectoryLoader.log`, and a definition of possible queue types.

Default values are set up in these files. You should not need to modify them.

Queue Manager Logs

When the Queue Manager is launched, `qdocDirectoryLoader.log` is created in:

```
TOMCAT_HOME/webapps/qxtendserver/WEB-INF/logs
```

This log tracks the Queue Manager and queues over a progression of reporting levels—error, warning, info, and debug.

The error level is controlled by modifying the level value for the logger name directoryloader in `qxtendlogging.xml`.

See "Log Report Levels" on page 53 for information on reporting levels.

Configuring the Transformation Engine

The transformation engine allows QXtend Inbound to accept non-QDoc XML files and apply an XSLT mapping specification to them to convert them to standard QDoc requests. Non-QDoc XML files are proprietary-format XML documents that are not in correct QDoc request format and that have a file extension other than `.req`.

The transformation engine polls for these documents in a queue that has been set up to allow non-standard XML documents. When one arrives, the transformation engine sends it to a Web service that extracts the message body, parses the data, and then transforms the data to a standard QDoc request. The transformation engine then creates a valid SOAP envelope for the document and outputs a QDoc request to the requests queue.

The engine also outputs an XML response document. This document will have an extension of `.mapok` or `.maperr` depending on the success or failure of the transformation processing. The original request document is also placed in this directory with an extension of `.`*`prp`*`_req`, where *`prp`* is the proprietary file extension.

The following sections provide instructions for creating a transformation queue for your proprietary-format XML documents.

Create an XSLT Mapping Specification

For a given proprietary XML document such as `syncCustomer.`*`prp`*, where `.`*`prp`* is the proprietary file extension, identify the target QDoc. Using the schemas for that QDoc, create an XSLT mapping specification to follow during transformation of a `.`*`prp`* file to the `.req` QDoc format. Store the mapping specification in:

```
TOMCAT_HOME/webapps/qxtendserver/WEB-INF/mappingSpecs
```

Although the file name can be anything, the mapping specification file name could include the document standard, a version number, the document type, and the destination; for example:

```
prpxml-001-synccust-eB2qdoc.xsl
```

In this case, `prpxml` is the document standard, `001` denotes the first version of the file, `synccust` is the document type, and `eB2qdoc` is the destination. The document standard is a value you create and assign for reference purposes in QXI configuration files.

Create a Request Parser

If your incoming XML document is in a proprietary format, the document type and version are required to determine the transformation stylesheet. The request parser extracts this. Because the location of these values in each proprietary file format differs, a new request parser is required.

To create a new request parser, follow these steps:

1. Create the following directory:

   ```
   WEB-INF/classes/com/qad/qxtend/queue/transformation
   ```

   This directory is required for the system to find your modified code before the QAD-supplied code.

2. Copy the existing request parser program:

   ```
   WEB-INF/RequestParser.java
   ```

3. Save the copied file in the directory created in step 1.

4. Modify it for your proprietary file format.

5. To configure the new request parser, add an entry to `requestparsermanager.xml` as shown in "requestparsermanager.xml Changes" on page 67.

6. Add the file extension mapping to the document standard used by the new request parser in `directoryloaderconfig.xml` as shown in "directoryloaderconfig.xml Changes" on page 67.

Modify QXtend Configuration Files

The transformation engine relies on several QXtend Inbound configuration files in order to call the correct transformation for the correct files. You must make minor modifications in an XML or text editor to the following files:

- `directoryloaderconfig.xml`
- `requestparsermanager.xml`
- `transformationmanager.xml`

directoryloaderconfig.xml Changes

In `directoryloaderconfig.xml`, add the new document standard:

```
<directoryloaderConfig>

...
    <documentStandards
        <documentStandard extension=”prp”>PRPXML
        </documentStandard>
    <documentStandards

...

<directoryloaderConfig>
```

requestparsermanager.xml Changes

Add the document standard to `requestparsermanager.xml`:

```
<requestParserManager>
    <requestParsers>
        <requestParser>
            <parserType>PRPXML</parserType>
            <parserClass>com.qad.common.queue.transformation.
            PRPXMLRequestParser</parserClass>
        </requestParser>
    </requestParsers>
</requestParserManager>
```

transformationmanager.xml Changes

Add the document standard as a contentTypeMap and the mapping specification including the mapping specification file name to transformationManager.xml:

```
<transformationManager>
    <transformers>
        <transformer>
            <transformerType>XML</transformerType>
            <transformerClass>com.qad.common.queue.
            transformation.XMLTransformer</transformerClass>
        </transformer>

        <contentTypeMaps>
            <contentTypeMap documentStandard=PRPXML</
            contentTypeMap>
        </contentTypeMaps>

        <mappingSpecifications>
            <mappingSpecification
            receiver=”eb2”
            documentStandard=”PRPXML”
            documentType=”syncCustomer”
            documentVersion=”001”>prpxml-001-synccust-
            eB2qdoc.xsl</mappingSpecification>
        </mappingSpecifications>
    </transformers>
</transformationManager>
```

This completes the configuration of the Queue Manager to accept non-standard XML documents. Test the new configuration thoroughly.

Starting Queue Manager

Access the Queue Manager through the QXtend Manager at:

```
http://<hostname>:<tomcat_port>/qxtendserver/
```

This assumes Tomcat and the target MFG/PRO instances are running.

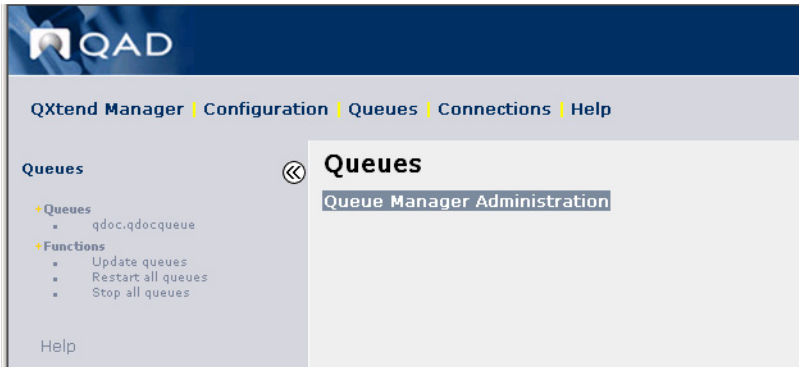


Fig. 5.2
Queue Manager Main Menu

From the main Queue Manager page, you can access individual queues, update or add queues, restart queues, and stop all queues.

Using the Queue Manager

In addition to configuring queues, you can stop or restart queues, view requests and responses in a queue, sort and filter those views, delete requests or responses, and you can edit and resubmit a failed request.

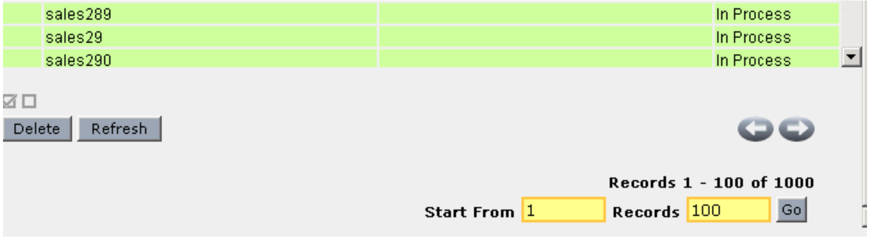
View Individual Queues

When you choose a queue name under Queues on the left-hand menu, or select a queue in the Functions|Update Queues screen, the contents of the queue display. The queue document list is color coded.

- Red lines are documents that returned an error.
- Blue are successful documents.
- Green are in process.

By default, 100 records are shown. Use the scrollbar on the queue to view the full 100 documents. You also can set the number of documents displayed using the Records settings at the bottom of the screen.

Fig. 5.3
Set Number of Documents in Queue to Display



	sales289		In Process
	sales29		In Process
	sales290		In Process

Records 1 - 100 of 1000

Start From 1 Records 100

Enter the starting and ending record number and choose Go to update the display. Entering 1 and 10 displays the first 10 records; entering 25 and 50 displays records 25 through 50.

Sorting

You can sort a queue by choosing the sort column name. Click once to sort in ascending order. Click again to sort descending.

Filtering

You can filter the queue by entering any criteria text in the field above the table. You then select the document type from the drop-down list. Choose Filter to select those queues that include the text and document type. The filtered result is case sensitive.

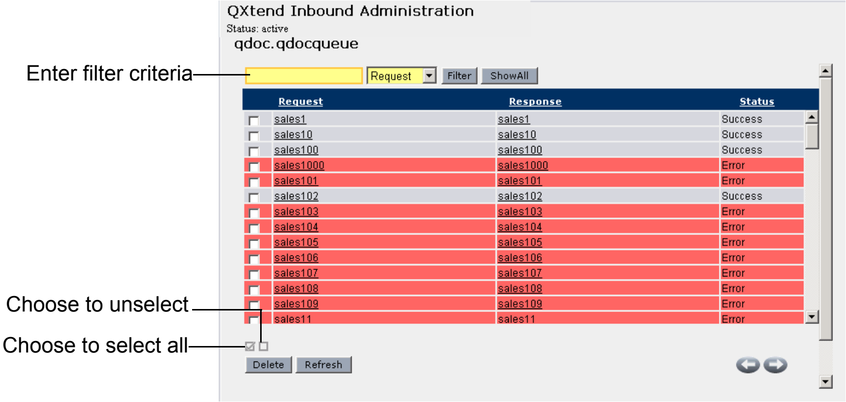
Example Enter *test* and select Warnings. Then choose Filter. This displays a list of the warnings present in all queues that contain test in the name.

Choose Show All to display all queues again.

Note To maintain the best possible performance, filter and sort options do not refresh the data; they filter data already displayed. To refresh the data with changes to the server, choose Refresh prior to sorting or filtering.

Deleting

Error, Warning, and Success status documents have a selection box next to them. You can choose in the box to select the document and then choose Delete to remove them from the queue.



Enter filter criteria

Choose to unselect

Choose to select all

QXtend Inbound Administration

Status: active

qdoc.qdocqueue

	Request	Response	Status
□	sales1	sales1	Success
□	sales10	sales10	Success
□	sales100	sales100	Success
□	sales1000	sales1000	Error
□	sales101	sales101	Error
□	sales102	sales102	Success
□	sales103	sales103	Error
□	sales104	sales104	Error
□	sales105	sales105	Error
□	sales106	sales106	Error
□	sales107	sales107	Error
□	sales108	sales108	Error
□	sales109	sales109	Error
□	sales11	sales11	Error

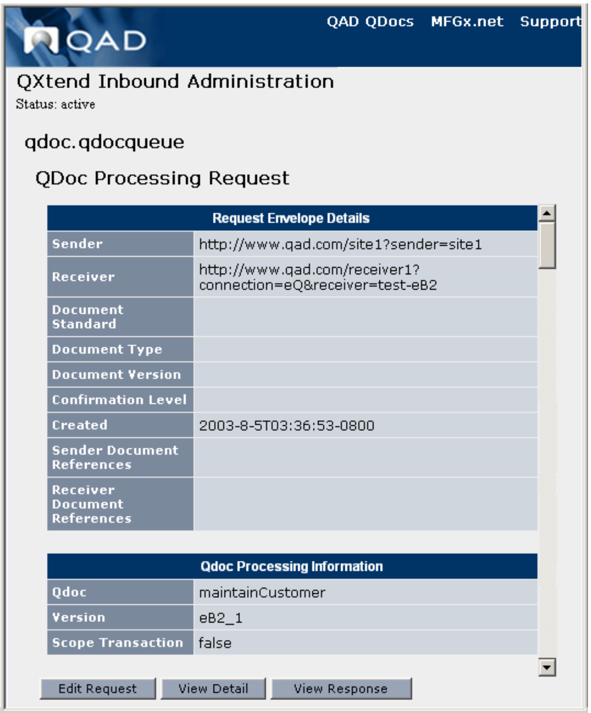
Fig. 5.4
Error and Success Documents in Queue

In the lower left-hand corner above the Delete button are select-all and select-none symbols. Choose the select-all symbol—the box with a checkmark inside—to select all documents in the queue; choose the empty box to unselect all documents.

Viewing Documents

For any request or response document, choose the document name in the queue display to view the document. A summary screen displays.

Fig. 5.5
Summary View of a Request Document



QXtend Inbound Administration

Status: active

qdoc.qdocqueue

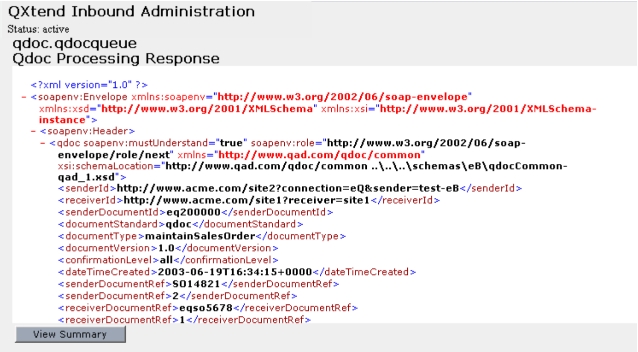
QDoc Processing Request

Request Envelope Details	
Sender	http://www.qad.com/site1?sender=site1
Receiver	http://www.qad.com/receiver1?connection=eQ&receiver=test-eB2
Document Standard	
Document Type	
Document Version	
Confirmation Level	
Created	2003-8-5T03:36:53-0800
Sender Document References	
Receiver Document References	

Qdoc Processing Information	
Qdoc	maintainCustomer
Version	eB2_1
Scope Transaction	false

Choose View Detail to view the XML content of the document.

Fig. 5.6
Detail View (XML) of Response Document



QXtend Inbound Administration

Status: active

qdoc.qdocqueue

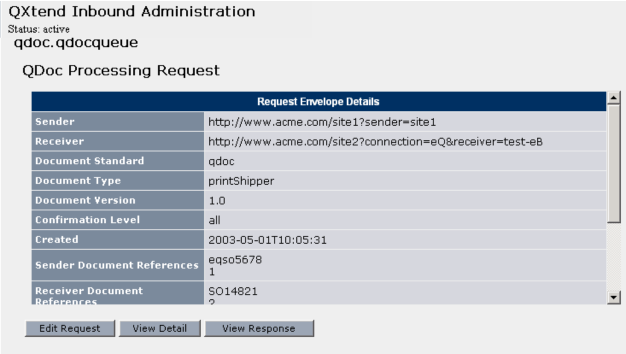
Qdoc Processing Response

```
  <?xml version="1.0" ?>
- <soapenv:Envelope xmlns:soapenv="http://www.w3.org/2002/06/soap-envelope"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance">
  - <soapenv:Header>
    - <qdoc soapenv:mustUnderstand="true" soapenv:role="http://www.w3.org/2002/06/soap-
        envelope/role/next" xmlns="http://www.qad.com/qdoc/common"
        xsi:schemaLocation="http://www.qad.com/qdoc/common ..\..\..\schemas\eB\qdocCommon-
        qad_1.xsd">
        <senderId>http://www.acme.com/site2?connection=eQ&sender=test-eB</senderId>
        <receiverId>http://www.acme.com/site1?receiver=site1</receiverId>
        <senderDocumentId>eq200000</senderDocumentId>
        <documentStandard>qdoc</documentStandard>
        <documentType>maintainSalesOrder</documentType>
        <documentVersion>1.0</documentVersion>
        <confirmationLevel>all</confirmationLevel>
        <dateTimeCreated>2003-06-19T16:34:15+0000</dateTimeCreated>
        <senderDocumentRef>SO14821</senderDocumentRef>
        <senderDocumentRef>2</senderDocumentRef>
        <receiverDocumentRef>eqso5678</receiverDocumentRef>
        <receiverDocumentRef>1</receiverDocumentRef>
```

Edit Failed Submissions

If you are viewing a request document with an Error status, you can display the document in an editing window. You can edit the XML and resubmit the document to the queue.

1 Select either the request or response document name in the queue. The summary view opens.



QXtend Inbound Administration

Status: active

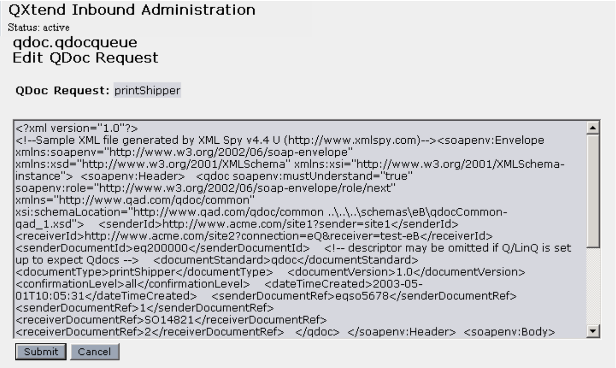
qdoc.qdocqueue

QDoc Processing Request

Request Envelope Details	
Sender	http://www.acme.com/site1?sender=site1
Receiver	http://www.acme.com/site2?connection=eQ&receiver=test-eB
Document Standard	qdoc
Document Type	printShipper
Document Version	1.0
Confirmation Level	all
Created	2003-05-01T10:05:31
Sender Document References	eqso5678 1
Receiver Document References	SO14821 2

Fig. 5.7 Summary View of an Error Status QDoc

2 Choose Edit Request to open the XML editor.



QXtend Inbound Administration

Status: active

qdoc.qdocqueue

Edit QDoc Request

QDoc Request: printShipper

```
<?xml version="1.0"?>
<!--Sample XML file generated by XML Spy v4.4 U (http://www.xmlspy.com)--><soapenv:Envelope
xmlns:soapenv="http://www.w3.org/2002/06/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">  <soapenv:Header>   <qdoc soapenv:mustUnderstand="true"
soapenv:role="http://www.w3.org/2002/06/soap-envelope/role/next"
xmlns="http://www.qad.com/qdoc/common"
xsi:schemaLocation="http://www.qad.com/qdoc/common ..\..\..\schemas\eB\qdocCommon-
qad_1.xsd">    <senderId>http://www.acme.com/site1?sender=site1</senderId>
<receiverId>http://www.acme.com/site2?connection=eQ&receiver=test-eB</receiverId>
<senderDocumentId>eq200000</senderDocumentId>    <!-- descriptor may be omitted if Q/LinQ is set
up to expect Qdocs -->    <documentStandard>qdoc</documentStandard>
<documentType>printShipper</documentType>    <documentVersion>1.0</documentVersion>
<confirmationLevel>all</confirmationLevel>    <dateTimeCreated>2003-05-
01T10:05:31</dateTimeCreated>    <senderDocumentRef>eqso5678</senderDocumentRef>
<senderDocumentRef>1</senderDocumentRef>
<receiverDocumentRef>SO14821</receiverDocumentRef>
<receiverDocumentRef>2</receiverDocumentRef>   </qdoc>  </soapenv:Header>  <soapenv:Body>
```

Fig. 5.8 XML Editor for Error Status QDocs

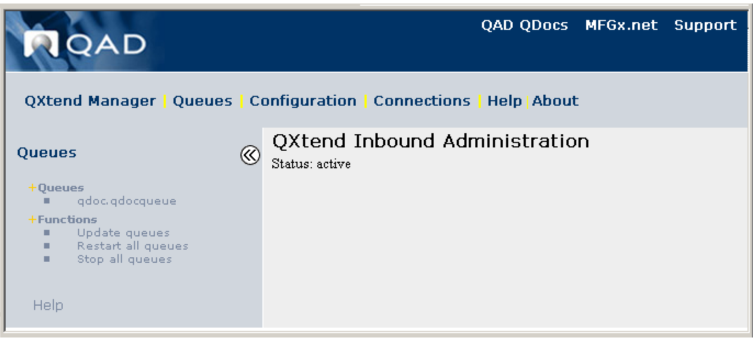
3 Select the QDoc text and make changes to the QDoc as required.

4 Choose Submit. The message "Request re-submitted successfully" displays. The document is immediately updated in the queue. Review the related response document for further error checking.

Configuring Queues

From any screen in the Queue Manager, you can select Functions on the left-hand menu. The functions are:

- Update queues.
- Restart all queues. This stops and restarts all queues.
- Stop all queues. This stops all queues.

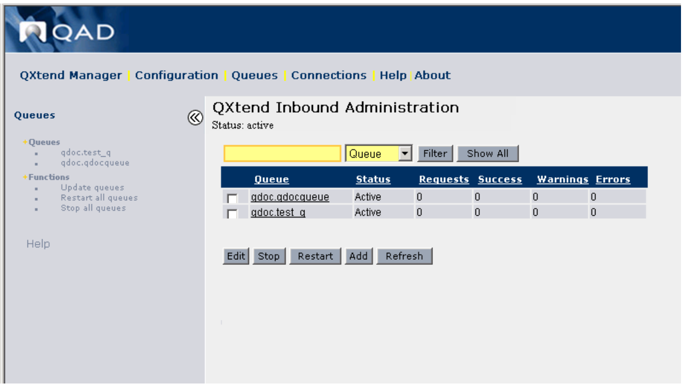


QXtend Inbound Administration

Status: active

Fig. 5.9
Queue Manager Functions Menu

When you select Update Queues under the Functions menu, a list of existing queues displays. You can filter and sort this list as described under “Sorting” and “Filtering” on page 70.



QXtend Inbound Administration

Status: active

	Queue	Status	Requests	Success	Warnings	Errors
□	qdoc.qdocqueue	Active	0	0	0	0
□	qdoc.test_q	Active	0	0	0	0

Fig. 5.10
Update Queue

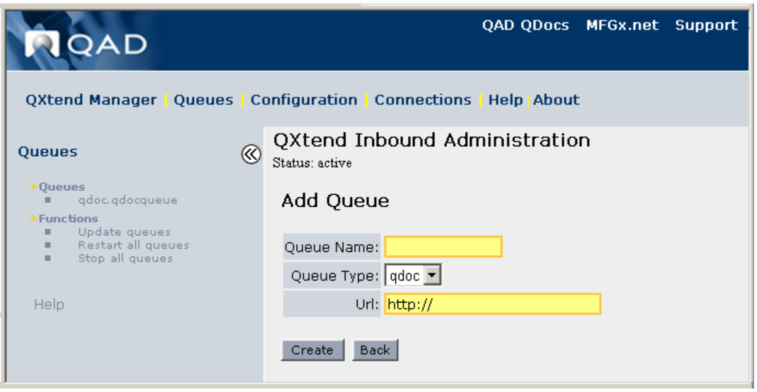
Stop or Restart Queues

1 To stop or restart one or more queues, choose the checkbox next to the queue names you want to stop or restart.

2 Choose Stop to stop the selected queues; choose Restart to shut down and restart them.

Add a Queue

1 To add a queue, choose Add. The Add Queue screen displays.



QXtend Inbound Administration

Status: active

Add Queue

Queue Name:

Queue Type: qdoc

Url: http://

Fig. 5.11
Add a Queue

2 Enter the following:

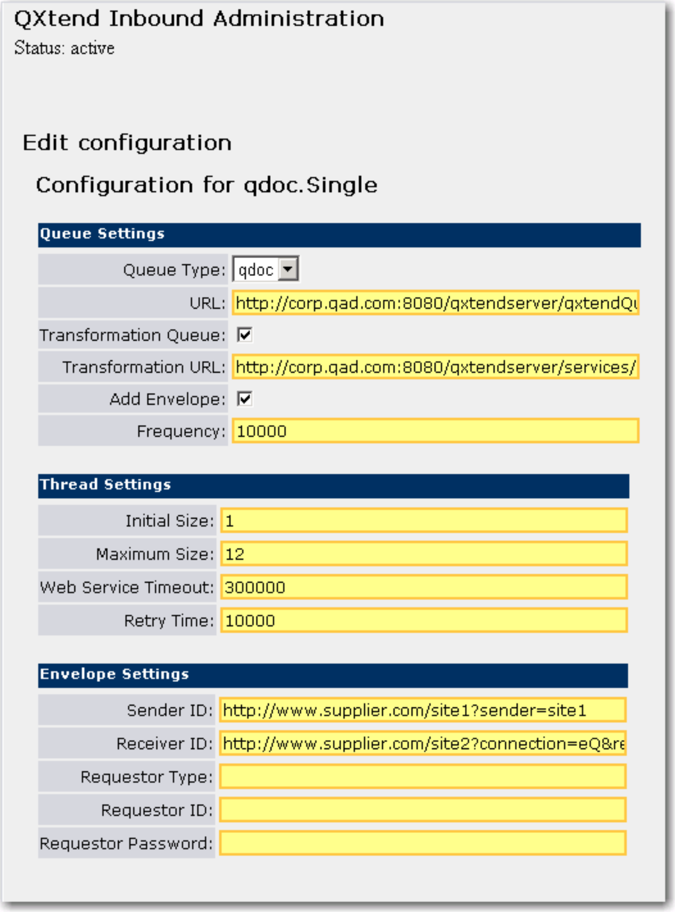
Queue Name. You can enter any name here. The final queue name will be `<queue type>.<your queue name>`. So if you enter `testQ` here, the queue name becomes `qdoc.testQ`.

Queue Type. Queue types identify the type of requests on the queue. QXI contains one queue type, `qdoc`.

URL. URL (Universal Resource Locator) is the location of the QXtend Inbound Web Services component.

3 Choose Create. The queue is created with a default configuration. The Edit Configuration page displays to enable you to edit the configuration.

Fig. 5.12
Edit Queue



QXtend Inbound Administration

Status: active

Edit configuration

Configuration for qdoc.Single

Queue Settings

Queue Type:	qdoc
URL:	http://corp.qad.com:8080/qxtendserver/qxtendQ
Transformation Queue:	☑
Transformation URL:	http://corp.qad.com:8080/qxtendserver/services/
Add Envelope:	☑
Frequency:	10000

Thread Settings

Initial Size:	1
Maximum Size:	12
Web Service Timeout:	300000
Retry Time:	10000

Envelope Settings

Sender ID:	http://www.supplier.com/site1?sender=site1
Receiver ID:	http://www.supplier.com/site2?connection=eQ&re
Requestor Type:	
Requestor ID:	
Requestor Password:	

Queue Settings

Update the fields using the details that follow:

Queue Type. Queue types identify the type of requests on the queue. QXI contains one queue type, qdoc.

URL. URL is the location of the QXtend Inbound Web Services component.

Transformation Queue. Determines whether this queue accepts non-standard XML documents for transformation.

Transformation URL. URL is the location of the QXtend Inbound Web Services component for the transformation engine.

Add Envelope. All incoming requests must include a valid SOAP envelope. If the incoming requests do not have a SOAP envelope, set this to true and the Queue Manager adds a SOAP envelope to each incoming message before sending it to QXtend Inbound. If Add Envelope is checked, enter values for Envelope Settings as well.

Frequency. In milliseconds, set the interval at which to poll the queue.

Envelope Settings

The following settings are only required if Add Envelope is checked in the Edit Configuration screen:

Sender ID. This identifies the QDoc source application. It is defined as a URL so that future QAD product releases can more easily use Internet-based industry standards for organizational identification.

Most of the URL is not yet used; however, at a minimum the Sender ID must include the `sender` parameter. To uniquely identify a sender, add a sender designation such as site:

```
sender=site1
```

Receiver ID. This identifies the QDoc recipient, which is by definition an MFG/PRO instance. It is defined as a URL so that future QAD product releases can more easily use Internet-based industry standards for organizational identification. Most of the URL is not yet used; however, at a minimum the Receiver ID must include the `receiver` parameter. To uniquely identify a receiver, add an MFG/PRO instance designation:

```
receiver=eB2_Prod
```

Requestor Type. QXI can validate incoming QDoc requests based on requestor type, ID, and password. Enter the requestor type as an alphanumeric string.

Requestor ID. The requestor ID is the user log-in from the requestor application. The requestor ID and password in a QDoc request are validated against the values stored for the queue. If the validation fails, the QDoc is returned with a SOAP error.

Requestor Password. The requestor password is the user password from the requestor application. The requestor password is validated along with the user ID.

Thread Settings

Use the Thread Settings to configure the queue as either a single or multi-threaded queue.

Initial Size. The starting number of threads. Set to 1 for single threaded queues. Set the size to at least 2 for multi-threaded queues.

Maximum Size. The maximum number of threads (or QDocs) that can be processed at one time. One thread is the equivalent of one MFG/PRO user.

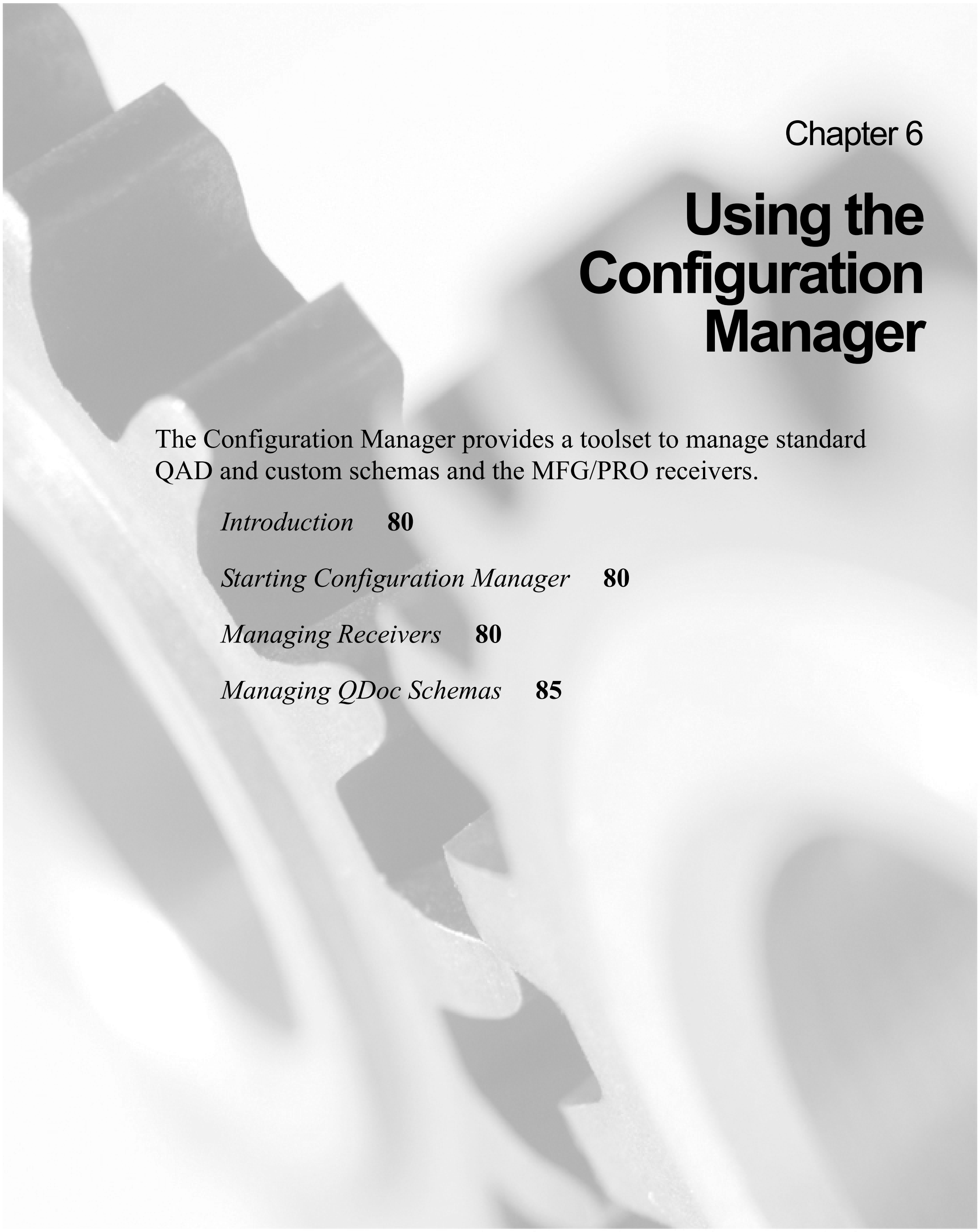
Web Service Timeout. The time-out in milliseconds for attempts to send QDocs to the QXtend Inbound Web service.

Retry Time. Interval in milliseconds between attempts if the Web service call fails.

4. When the queue setup is complete, choose Submit. The message “Queue Config saved successfully” displays.

Modify a Queue

1. To update a queue configuration, select the checkbox next to the queue you want to edit.
2. Choose Edit. The Edit Configuration screen displays. Make changes as described in step 3 under “Add a Queue” on page 75.
3. When the queue setup is complete, choose Submit. The message “Queue Config saved successfully” displays.



Chapter 6

Using the Configuration Manager

The Configuration Manager provides a toolset to manage standard QAD and custom schemas and the MFG/PRO receivers.

Introduction

The Configuration Manager provides an interface to manage QDoc schemas, create and modify MFG/PRO receivers, and allocate QDoc schemas to the receivers.

Starting Configuration Manager

For information on configuration errors, see “Configuration Exceptions” on page 260.

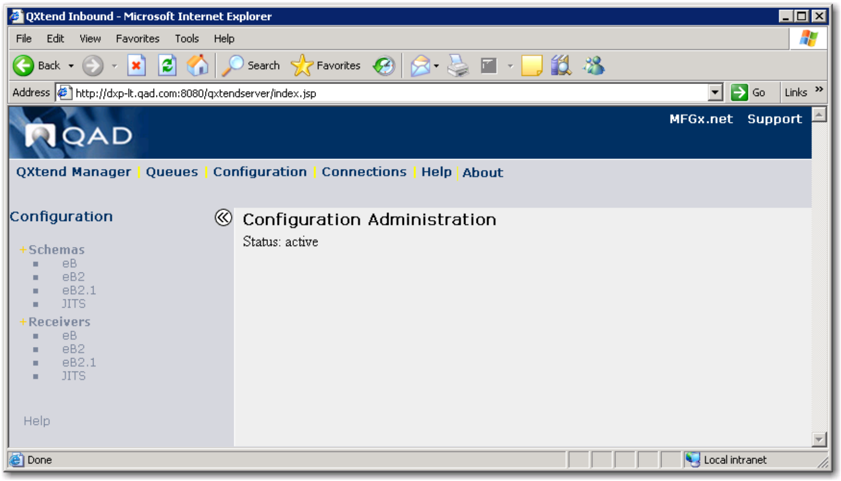
Access the Configuration Manager through the QXtend Manager at:

```
http://<hostname>:<tomcat_port>/qxtendserver/
```

This assumes Tomcat and the target MFG/PRO instances are running.

The Configuration Manager is launched from the QXtend Inbound interface and lets you manage your master list of available QDoc schemas and set up and configure your MFG/PRO receivers.

Fig. 6.1 Configuration Manager Interface



QXtend Inbound - Microsoft Internet Explorer

Address http://dxp-lt.qad.com:8080/qxtendserver/index.jsp

Configuration Administration

Status: active

Managing Receivers

Receivers are named MFG/PRO instances. You can define a single MFG/PRO instance for all your inbound QDocs, or multiple receivers to support test and training databases, multiple MFG/PRO versions, or multiple databases.

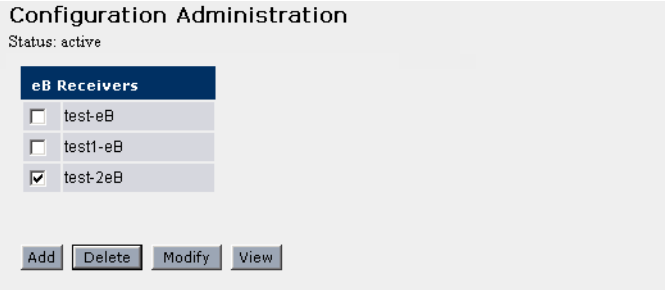
In the Configuration Manager, you can:

- Add receivers.
- View a list of receivers.
- Add standard or custom schemas to receivers.
- Remove standard or custom schemas from receivers.
- Delete receivers.

Add a Receiver

1 Launch the Receiver interface by choosing Receivers.

2 The currently available receivers display.



Configuration Administration

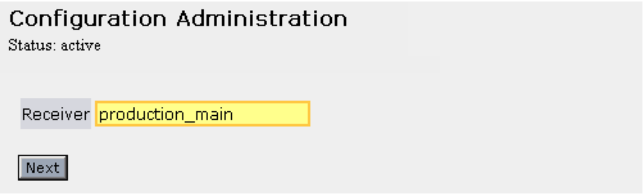
Status: active

eB Receivers

- □ test-eB
- □ test1-eB
- ☑ test-2eB

Fig. 6.2
Receiver View in Configuration Manager

3 To add a receiver, choose Add. If QXI is active, the suspend options display. Choose a suspend option to cancel the add or continue. The Add Receiver screen displays.



Configuration Administration

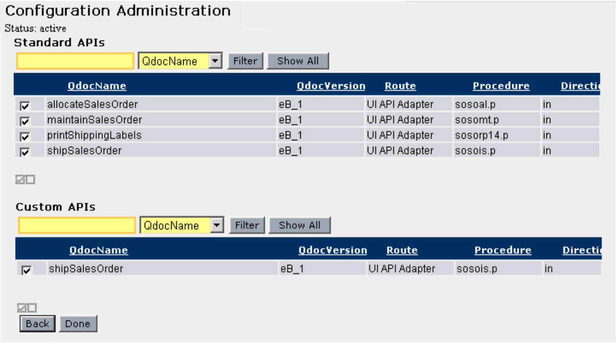
Status: active

Receiver: production_main

Fig. 6.3
Naming a Receiver

4 Enter a name for the receiver. Choose Next. The list of available QDoc APIs displays.

Fig. 6.4
Adding QDoc APIs to a New Receiver



Configuration Administration

Status: active

Standard APIs

	QdocName	QdocVersion	Route	Procedure	Directi
☑	allocateSalesOrder	eB_1	UI API Adapter	sosoal.p	in
☑	maintainSalesOrder	eB_1	UI API Adapter	sosomt.p	in
☑	printShippingLabels	eB_1	UI API Adapter	sosorp14.p	in
☑	shipSalesOrder	eB_1	UI API Adapter	sosois.p	in

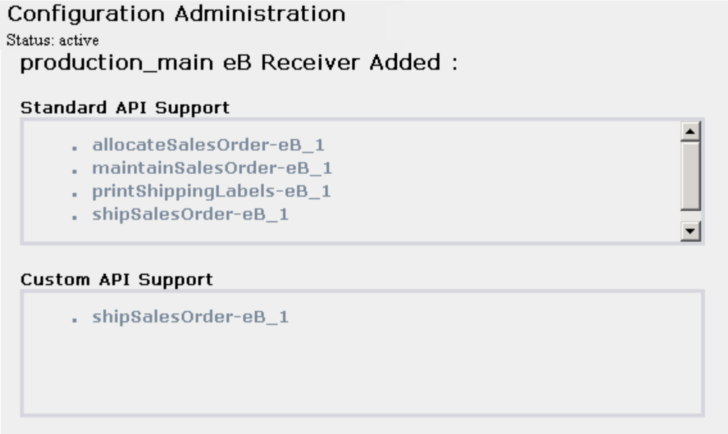
Custom APIs

	QdocName	QdocVersion	Route	Procedure	Directi
☑	shipSalesOrder	eB_1	UI API Adapter	sosois.p	in

5 Select from the available lists of standard and custom QDocs by clicking the QDocs you want. Choose Done to make the assignment. A confirmation screen displays.

If you suspended QXI in step 3, you are given the option of resuming it. If you choose to do so, QXI services are restored and a confirmation message displays.

Fig. 6.5
QDocs Added to a New Receiver



Configuration Administration

Status: active

production_main eB Receiver Added :

Standard API Support

- allocateSalesOrder-eB_1
- maintainSalesOrder-eB_1
- printShippingLabels-eB_1
- shipSalesOrder-eB_1

Custom API Support

- shipSalesOrder-eB_1

6 To exit the add new receiver confirmation screen or the resume QXI services confirmation, choose any other menu option in the QXtend Manager interface.

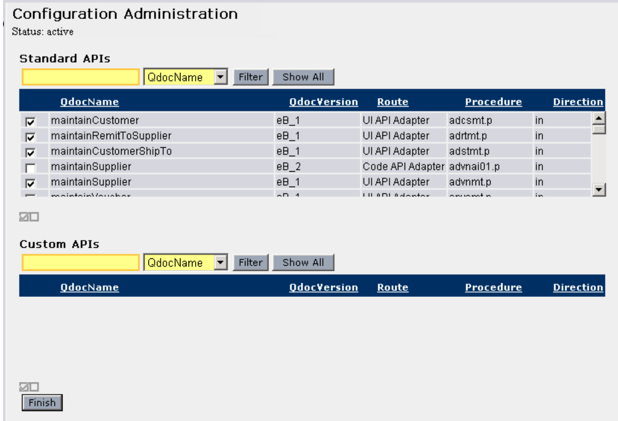
Add a Schema to an Existing Receiver

1. Launch the Receiver interface by clicking Receivers. The currently available receivers display.

See “Managing QDoc Schemas” on page 85.

2. To add a schema to a receiver, select the receiver by clicking in the box next to it. Choose Modify and then choose the appropriate Suspend option if QXI is active. The list of QDocs currently assigned to the receiver displays.

3. Choose Add QDoc. The list of standard and custom QDocs not currently assigned to the receiver displays.



Configuration Administration

Status: active

Standard APIs

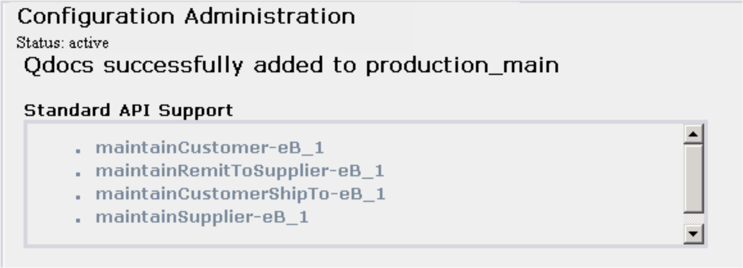
	QdocName	QdocVersion	Route	Procedure	Direction
☑	maintainCustomer	eB_1	UI API Adapter	adcsmt.p	in
☑	maintainRemitToSupplier	eB_1	UI API Adapter	adrtmt.p	in
☑	maintainCustomerShipTo	eB_1	UI API Adapter	adstmt.p	in
☐	maintainSupplier	eB_2	Code API Adapter	advnai01.p	in
☑	maintainSupplier	eB_1	UI API Adapter	advnmt.p	in

Custom APIs

	QdocName	QdocVersion	Route	Procedure	Direction

Fig. 6.6
Adding a QDoc to an Existing Receiver

4. Select the QDocs you want to add and choose Finish. A confirmation displays. If you suspended QXI in step 2, you are given the option of resuming it. If you choose to do so, QXI services are restored and a confirmation message displays.



Configuration Administration

Status: active

Qdocs successfully added to production_main

Standard API Support

- maintainCustomer-eB_1
- maintainRemitToSupplier-eB_1
- maintainCustomerShipTo-eB_1
- maintainSupplier-eB_1

Fig. 6.7
QDocs Added to an Existing Receiver

5 To exit the add schema to a receiver confirmation screen or the resume QXI services confirmation, choose any other menu option in the QXtend Manager interface.

Remove a Schema from an Existing Receiver

1 Launch the Receiver interface by choosing Receivers. The currently available receivers display.

2 To remove a schema from a receiver, select the receiver by clicking in the box next to it. Choose Modify and then choose the appropriate Suspend option if QXI is active. The list of QDocs currently assigned to the receiver displays.

3 Select the QDocs you want to remove from the receiver by clicking in the boxes next to them. Choose Remove QDoc.

4 A confirmation screen displays. If you suspended QXI in step 2, you are given the option of resuming it. If you choose to do so, QXI services are restored and a confirmation message displays.

5 To exit the remove schema confirmation screen or the resume QXI services confirmation, choose any other menu option in the QXtend Manager interface.

Delete a Receiver

1 Launch the Receiver interface by clicking Receivers. The currently available receivers display.

2 Select the receiver you want to delete by clicking in the box next to it. Choose Delete.

3 You are asked, “Are you sure you want to delete the receiver?” Choose OK.

4 You are asked, “Are all files within the receiver closed?” This should be true if no documents are in transit for the receiver. Choose OK. If QXI is active, the suspend options display. Choose a suspend option to cancel the add or continue.

5 A confirmation screen displays. The deleted receiver is backed up to:

```
TOMCAT_HOME/webapps/qxtendserver/WEB-INF/
deletedReceivers
```

6 If you suspended QXI services in step 4, you have the option of resuming QXtend Manager at this point. If you choose to do this, QXI returns to the active state and a confirmation displays.

7 To exit the delete receiver confirmation screen or the resume QXI services confirmation, choose any other menu option in the QXtend Manager interface.

Managing QDoc Schemas

The Schemas option controls your master list of QDoc schemas. This option display a list of available schemas in the global descriptor files located in:

```
TOMCAT_HOME/webapps/qxtendserver/WEB-INF/descriptors/
<mfgpro-version>
```

The `qadQdocs.xml` file in this directory contains a list of all standard QDoc APIs for the specific MFG/PRO version. The `implementationQdocs.xml` file contains the list of all custom QDocs added through the Configuration Manager for the MFG/PRO version.

You can view, filter, and sort the lists of both standard and custom QDocs.

Multiple Events Files

You can also maintain multiple events files for a given schema. By default, events files are named like the menu-level MFG/PRO programs with an `.xml` extension. For example, the menu-level program `sosomt.p` would have a default events file of `sosomt.xml`.

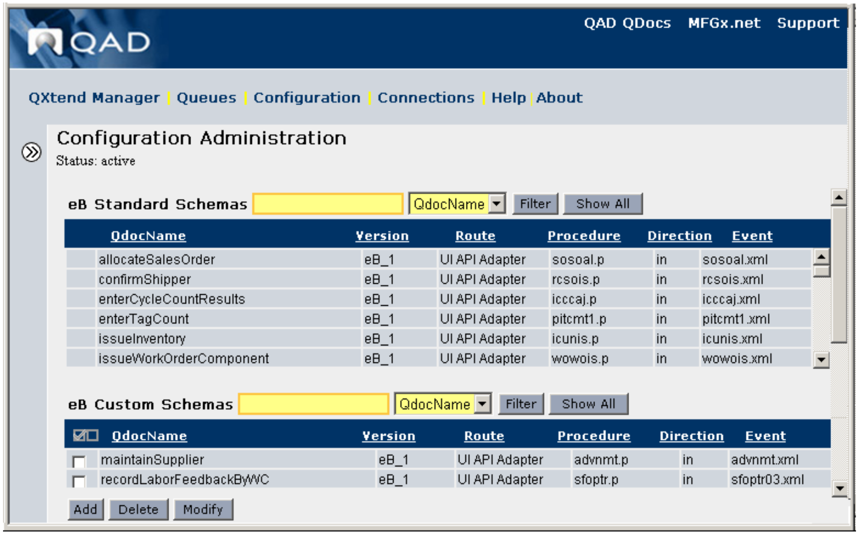
You can create multiple events files to support different paths through a program, as for example, when a site is using advanced pricing for some orders and not for others.

The ability to use different events files can also help you identify and correct performance problems for a given menu-level program.

View All QDocs by MFG/PRO Version

1 In the Configuration Manager, choose Schemas|<*mfgpro-version*>, where <*mfgpro-version*> is eB, eB2, or eB2.1. The master list of all available standard and custom schemas displays.

Fig. 6.8
Schema View in Configuration Manager



Configuration Administration
Status: active

eB Standard Schemas [ ] QdocName

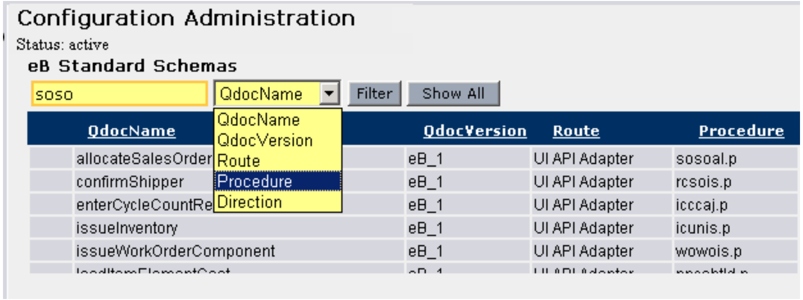
QdocName	Version	Route	Procedure	Direction	Event
allocateSalesOrder	eB_1	UI API Adapter	sosoal.p	in	sosoal.xml
confirmShipper	eB_1	UI API Adapter	rcsois.p	in	rcsois.xml
enterCycleCountResults	eB_1	UI API Adapter	icccaj.p	in	icccaj.xml
enterTagCount	eB_1	UI API Adapter	pitcmt1.p	in	pitcmt1.xml
issueInventory	eB_1	UI API Adapter	icunis.p	in	icunis.xml
issueWorkOrderComponent	eB_1	UI API Adapter	wowois.p	in	wowois.xml

eB Custom Schemas [ ] QdocName

☑☐	QdocName	Version	Route	Procedure	Direction	Event
☐	maintainSupplier	eB_1	UI API Adapter	advnmt.p	in	advnmt.xml
☐	recordLaborFeedbackByWC	eB_1	UI API Adapter	sfoptr.p	in	sfoptr03.xml

2 To limit the schema list by a text string criteria—such as all QDocs that contain "soso" in the procedure name—enter the text string in the field above the list and select the column to filter from the drop-down list.

Fig. 6.9
Entering Filter Criteria for Schemas

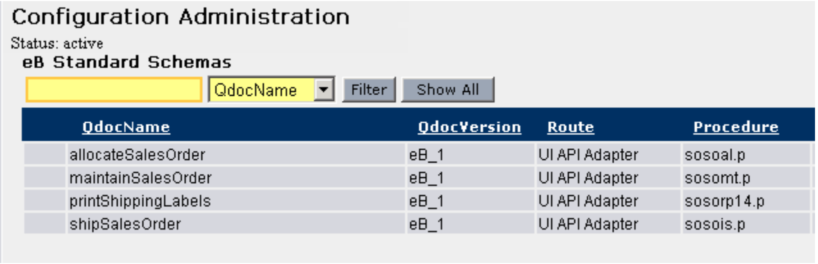


Configuration Administration
Status: active
eB Standard Schemas

soso QdocName

QdocName	QdocVersion	Route	Procedure
allocateSalesOrder	eB_1	UI API Adapter	sosoal.p
confirmShipper	eB_1	UI API Adapter	rcsois.p
enterCycleCountRe	eB_1	UI API Adapter	icccaj.p
issueInventory	eB_1	UI API Adapter	icunis.p
issueWorkOrderComponent	eB_1	UI API Adapter	wowois.p

3 Choose Filter to see the results. Choose Show All to clear the filter and view all documents.



Configuration Administration

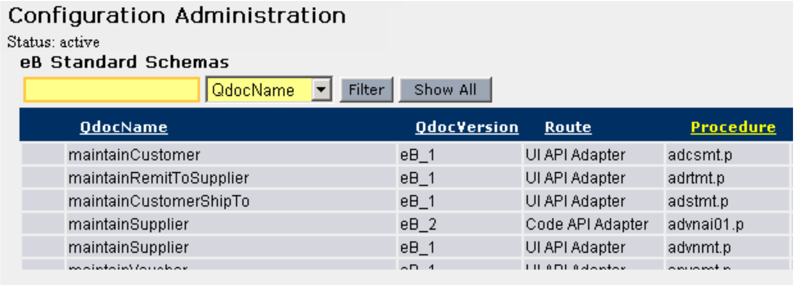
Status: active

eB Standard Schemas

QdocName	QdocVersion	Route	Procedure
allocateSalesOrder	eB_1	UI API Adapter	sosoal.p
maintainSalesOrder	eB_1	UI API Adapter	sosomt.p
printShippingLabels	eB_1	UI API Adapter	sosorp14.p
shipSalesOrder	eB_1	UI API Adapter	sosois.p

Fig. 6.10 Filtered Schema Results

4 To sort the contents of the schema list, click the column name of the column you want to sort by. The first time you click, the list is sorted in ascending order by the selected column. The second time you click the same column, the sort is in descending order.



Configuration Administration

Status: active

eB Standard Schemas

QdocName	QdocVersion	Route	Procedure
maintainCustomer	eB_1	UI API Adapter	adcsmt.p
maintainRemitToSupplier	eB_1	UI API Adapter	adrtmt.p
maintainCustomerShipTo	eB_1	UI API Adapter	adstmt.p
maintainSupplier	eB_2	Code API Adapter	advnai01.p
maintainSupplier	eB_1	UI API Adapter	advnmt.p

Fig. 6.11 Sorting Schemas

5 To exit the schema view, choose any other menu option in the QXtend Manager interface.

Add a Schema to the Master Lists

Schemas are the QDoc schemas and events files generated using QGen. QAD ships standard QDoc APIs. Additional standard QDocs may be made available on the QAD Web site at:

http://support.qad.com/

In addition, you can generate schemas and events files for customized or custom Progress programs.

For information on QGen, see “Using QGen” on page 111.

When you add the schemas through the Configuration Manager, new standard QAD schemas are added to:

```
TOMCAT_HOME/webapps/qxtendserver/WEB-INF/descriptors/
<mfgpro-version>/qadQdocs.xml
```

Custom schemas are added to:

```
TOMCAT_HOME/webapps/qxtendserver/WEB-INF/descriptors/
<mfgpro-version>/implementationQdocs.xml
```

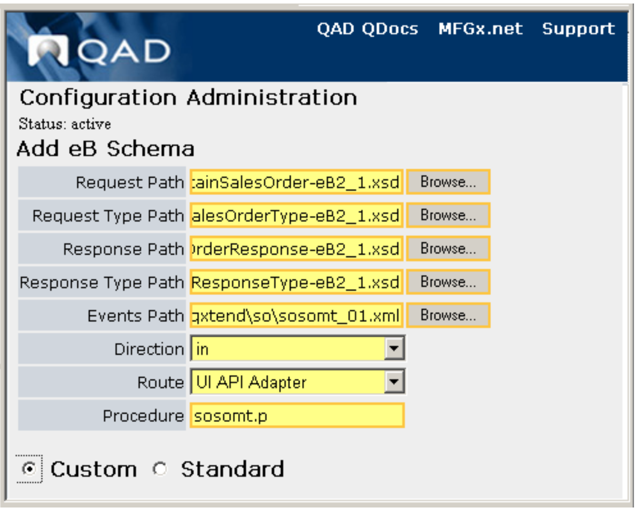
During the addition of the new QDocs, you can select which receivers to add them to, and the new QDocs are available to be added to any new receiver you create in the future.

You can add or modify QDocs, or overwrite existing QDocs. You can also delete custom QDocs.

1 To add a schema, choose Schemas|<*mfgpro-version*>. The master list of all available standard and custom schemas displays.

2 Choose Add at the bottom of the screen and then choose the appropriate Suspend option if QXI is active. The Add Schema screen displays.

Fig. 6.12
Adding a Schema



Configuration Administration

Status: active

Add eB Schema

Request Path	:ainSalesOrder-eB2_1.xsd	Browse...
Request Type Path	alesOrderType-eB2_1.xsd	Browse...
Response Path	›rderResponse-eB2_1.xsd	Browse...
Response Type Path	ResponseType-eB2_1.xsd	Browse...
Events Path	ɑxtend\so\sosomt_01.xml	Browse...
Direction	in	
Route	UI API Adapter	
Procedure	sosomt.p	

● Custom ○ Standard

3 Enter the following information:

Request Path. Specify the full path to the base request schema that is to be uploaded to the standard or custom schema directory in Tomcat; for example:

```
downloads/shipSalesOrder-eB_1.xsd
```

Request Type Path. Specify the full path to the request type schema, typically the same path as for the base schema; for example:

```
downloads/shipSalesOrderType-eB_1.xsd
```

Response Path. Optional. This path and file name are required only when you want to specify a response schema. Enter the path to the base response schema that is to be uploaded to the standard or custom schema directory in Tomcat; for example:

```
downloads/shipSalesOrderResponse-eB_1.xsd
```

Response Type Path. Optional. This path and file name are required only when you want to specify a response type schema. Enter the path to the base response type schema that is to be uploaded to the standard or custom schema directory in Tomcat; for example:

```
downloads/shipSalesOrderResponseType-eB_1.xsd
```

Events Path. Specify the full path and file name of the events file. This is the location where the events for the schema are saved, usually with the same name as the MFG/PRO source procedure, but with an `.xml` extension. The default path is:

```
TOMCAT_HOME/webapps/qxtendserver/WEB-INF/events/
mfgpro_version
```

To use a different version of the events file, specify the path to it here; for example:

```
TOMCAT_HOME/webapps/qxtendserver/WEB-INF/events/eb2/
downloads/sosois_02.xml
```

Direction. In or Out. Currently only incoming documents are supported.

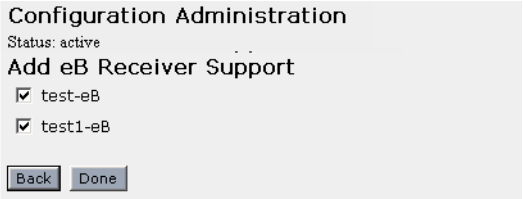
Route. Indicate the name of the adapter for this QDoc—UI API or code API.

Procedure. Specify the target procedure name in MFG/PRO, such as `customProgram.p` or `sosois.p`.

Choose the Custom radio button to install a custom QDoc; the Standard button to install a standard QDoc.

4 Choose Next. If the QDoc already exists, you are given the option to overwrite the existing QDoc. If you overwrite, you continue and the Add Receiver Support page displays. If you do not overwrite, you are returned to step 2 and must re-enter the information.

Fig. 6.13 Adding a Schema to Existing Receivers


Configuration Administration

Status: active

Add eB Receiver Support

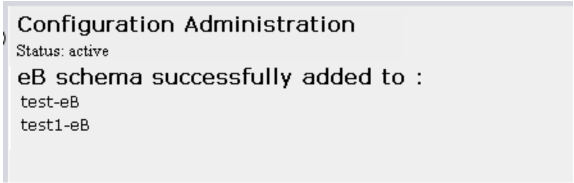
☑ test-eB

☑ test1-eB

5 If you want to add the new schema to receivers, select the receivers you want to add the new schema to and choose Done. A confirmation message displays.

6 If you suspended QXI services in step 2, you are given the option to resume. If you do so, QXI services resume and a confirmation message displays.

Fig. 6.14 Confirming a Schema Addition


Configuration Administration

Status: active

eB schema successfully added to :

test-eB

test1-eB

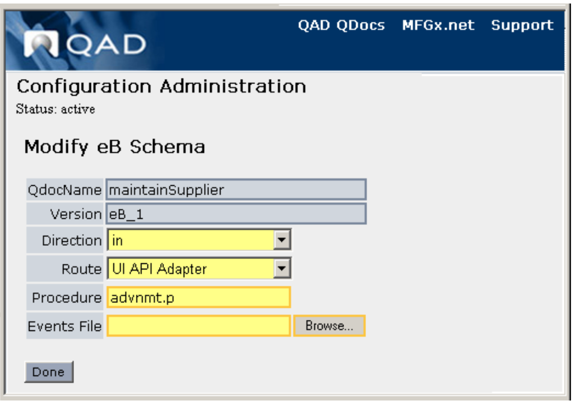
7 To exit the add schema or the resume QXI services confirmation screens, choose any other menu option in the QXtend Manager interface.

Modify a Schema Configuration

Once a schema has been added, you can modify a limited set of attributes.

1 To modify a schema configuration, choose Schemas|<*mfgpro-version*>. The master list of all available standard and custom schemas displays.

2 Select the schema you want to modify and choose Modify and then choose the appropriate Suspend option if QXI is active. The Modify Schema page displays.



Configuration Administration

Status: active

Modify eB Schema

QdocName	maintainSupplier
Version	eB_1
Direction	in
Route	UI API Adapter
Procedure	advnmt.p
Events File	

Fig. 6.15 Modify a Schema Configuration

3 Update Direction, Route, Procedure, or the Events File location and choose Done. A confirmation message displays.

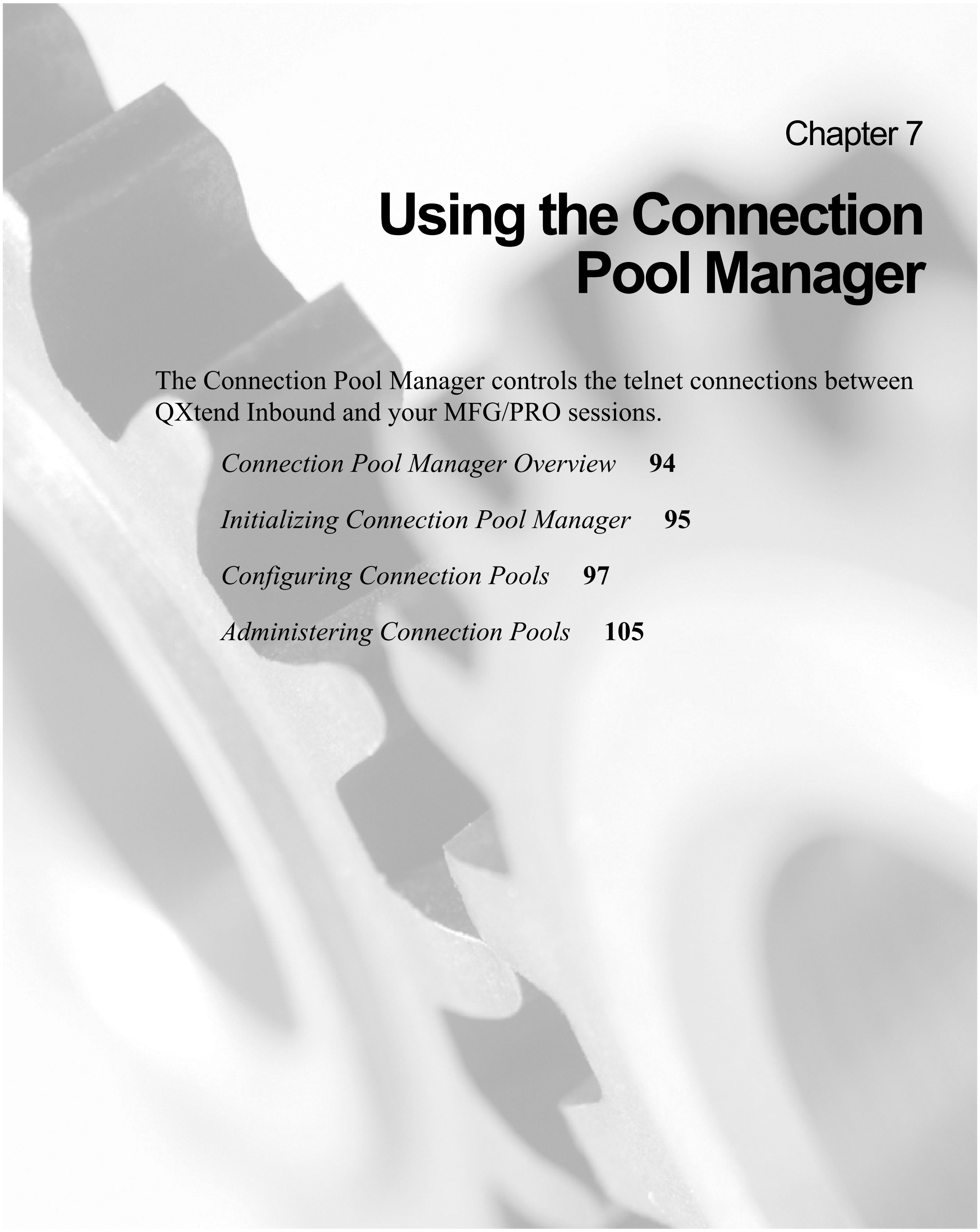
4 If you suspended QXI services in step 2, you are given the option to resume. If you do so, QXI services resume and a confirmation message displays.

5 To exit the modify schema or the resume QXI services confirmation screens, choose any other menu option in the QXtend Manager interface.

Delete a Custom Schema Configuration

You can delete a custom schema from the master list. It is removed from the appropriate `implementationQdocs.xml`. The associated events file is also deleted. You cannot delete standard QAD schemas.

1. To delete a custom schema configuration, choose Schemas|<*mfgpro-version*>. The master list of all available standard and custom schemas displays.
2. Select the custom schema you want to delete and choose Delete and then choose the appropriate Suspend option if QXI is active.
3. You are asked to confirm. Choose OK. A confirmation displays.
4. If you suspended QXI services in step 2, you are given the option to resume. If you do so, QXI services resume and a confirmation message displays.
5. To exit the delete custom schema or the resume QXI services confirmation screens, choose any other menu option in the QXtend Manager interface.



Chapter 7

Using the Connection Pool Manager

The Connection Pool Manager controls the telnet connections between QXtend Inbound and your MFG/PRO sessions.

Connection Pool Manager Overview

The Connection Pool Manager controls telnet connection pools between QXtend Inbound and your MFG/PRO and other application sessions, such as JIT/S. Each connection pool is identified by the type of user it serves—Desktop, UI API, JIT/S, or code API—and by a pool name, the host machine and port, and a system user.

Note Although Desktop appears as a connection type, do not maintain Desktop connections in the QXtend Connection Pool Manager.

Each connection pool consists of a minimum and maximum number of connections, and supports several timeout checkpoints. When QXtend Inbound starts, the active connection pools are started automatically. A pool with a minimum of five and a maximum of fifteen connections automatically creates the first five connections and keeps them idle awaiting connection requests from the identified user.

When a request arrives, one of the five connections is handed to the requester. That connection moves from the idle state to allocated and then to busy. The Connection Pool Manager then initializes another connection so that, as long as possible within the maximum, there are always five open connections available to incoming requests.

Each incoming QXI request specifies a receiver—a specific MFG/PRO instance. You define one API connection pool for each receiver—the pool name is the same as the receiver name. As requests arrive, the receiver is matched with the appropriate connection pool, and a connection is assigned for the transmission of a QDoc.

Connection Pool Manager Interface

The Connection Pool Manager interface lets you:

- Add and configure a new QXI connection pool.
- View, close, restart, modify, or delete a connection pool.
- View and manage current connections by state.
- View and manage current user sessions.

Overview

The Connection Pool Manager provides the following capabilities:

- Initializing Connection Pool Manager
- Configuring Connection Pools
- Administering Connection Pools

For information on Connection Pool errors, see “Connection Exceptions” on page 247.

Initializing Connection Pool Manager

Prior to using the Connection Pool Manager, you must create connection pools for each QXtend Inbound receiver.

Start Connection Pool Manager

Typically, you access the Connection Pool Manager through the QXtend Manager at:

```
http://<hostname>:<tomcat_port>/qxtendserver/
```

You can also access the Connection Pool Manager directly at:

```
http://<hostname>:<tomcat_port>/qxtendserver/
ConnectionManager.html/
```

Both methods assume Tomcat and the target MFG/PRO instances are running.

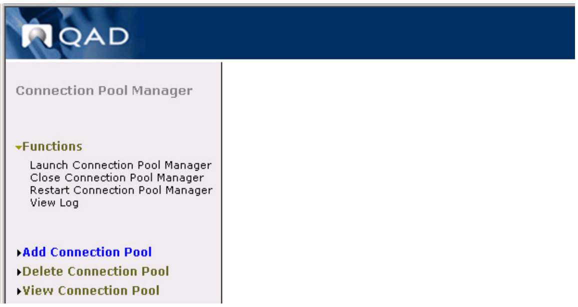
Administer the Connection Pool Manager

The Connection Pool Manager is launched from the QXtend Manager interface. It lets you restart and stop the manager itself, create and configure connection pools, and manage user sessions. The operations to control the Connection Pool Manager itself are launching, closing, and restarting the Connection Pool Manager, and viewing the Connection Pool Manager log.

1. In QXtend Manager, choose Connections. The Connection Pool Manager displays.

2. Choose the Functions menu. The Connection Pool Manager administrative options display.

Fig. 7.1 Connection Pool Manager Admin Functions Menu



3 Choose the following menu options to accomplish administrative tasks:

Launch Connection Pool Manager. This option starts the Connection Pool Manager after a close command.

Close Connection Pool Manager. This option closes all connection pools (not the Manager). All open connections and connection pools are closed; the Connection Pool Manager remains open.

Restart Connection Pool Manager. This option closes all connection pools and restarts the Connection Pool Manager.

View Log. This option displays the Connection Pool Manager log. See the next section for full details.

View the Connection Pool Log

When a connection pool is launched, `connectionPools.log` is created in:

```
TOMCAT_HOME/webapps/qxtendserver/WEB-INF/logs
```

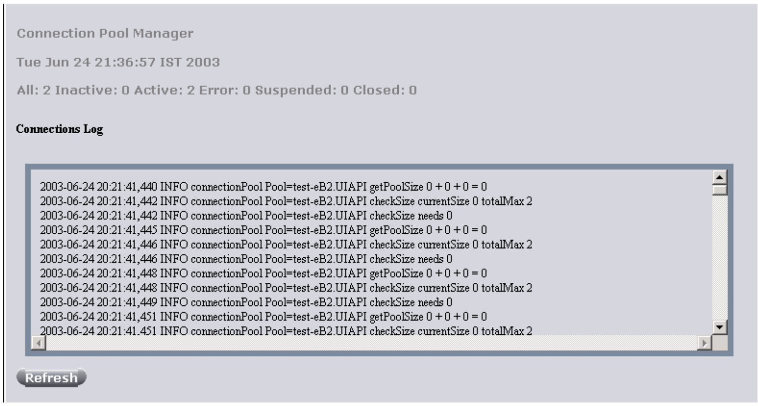
This log tracks the Connection Pool Manager and connection pools over a progression of reporting levels—error, warning, info, and debug.

See “Log Report Levels” on page 53 for information on reporting levels.

The error level is controlled by modifying the level value for the logger name connectionPool in `qxtendlogging.xml`. By default, the level is set to Info.

To view the connection pool log in Connection Pool Manager:

1 From the Connection Pool Manager Functions menu, choose View Log. The log displays.



Connection Pool Manager

Tue Jun 24 21:36:57 IST 2003

All: 2 Inactive: 0 Active: 2 Error: 0 Suspended: 0 Closed: 0

Connections Log

```
2003-06-24 20:21:41,440 INFO connectionPool Pool=test-eB2.UIAPI getPoolSize 0 + 0 + 0 = 0
2003-06-24 20:21:41,442 INFO connectionPool Pool=test-eB2.UIAPI checkSize currentSize 0 totalMax 2
2003-06-24 20:21:41,442 INFO connectionPool Pool=test-eB2.UIAPI checkSize needs 0
2003-06-24 20:21:41,445 INFO connectionPool Pool=test-eB2.UIAPI getPoolSize 0 + 0 + 0 = 0
2003-06-24 20:21:41,446 INFO connectionPool Pool=test-eB2.UIAPI checkSize currentSize 0 totalMax 2
2003-06-24 20:21:41,446 INFO connectionPool Pool=test-eB2.UIAPI checkSize needs 0
2003-06-24 20:21:41,448 INFO connectionPool Pool=test-eB2.UIAPI getPoolSize 0 + 0 + 0 = 0
2003-06-24 20:21:41,448 INFO connectionPool Pool=test-eB2.UIAPI checkSize currentSize 0 totalMax 2
2003-06-24 20:21:41,449 INFO connectionPool Pool=test-eB2.UIAPI checkSize needs 0
2003-06-24 20:21:41,451 INFO connectionPool Pool=test-eB2.UIAPI getPoolSize 0 + 0 + 0 = 0
2003-06-24 20:21:41.451 INFO connectionPool Pool=test-eB2.UIAPI checkSize currentSize 0 totalMax 2
```

Fig. 7.2
View Connection Pool Log

2 Scroll down to view the most recent entries.

Configuring Connection Pools

Prior to accepting any user requests from QXI, configure the necessary connection pools. This is accomplished in the Connection Pool Manager.

Add a Connection Pool

You add new connection pools in the Connection Pool Manager by the type of user the pool is going to support. The options are UI API, JIT/S, or code API pools. You name each pool with the receiver name of the target MFG/PRO instance and identify the pool by host machine and port. You then create a startup script for the pool telnet session, and enter the pool parameters.

When you create or modify a connection pool, it is created or modified in your `connectionManagerConfig.xml` file. The default version of this file is located in:

```
TOMCAT_HOME/webapps/qxtendserver/WEB-INF/conf/defaults
```

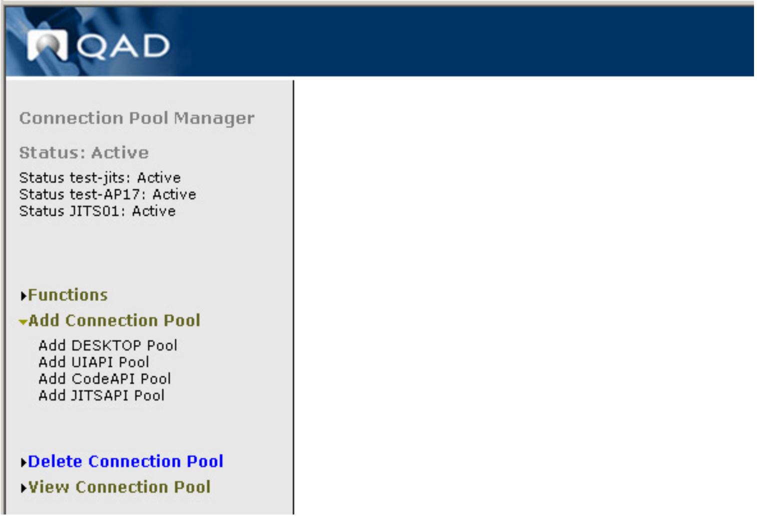
The server-specific version is stored one level up in the `/conf` directory.

Warning The `connectionManagerConfig.xml` file should not be modified manually. Make changes only through the interface.

Adding a UI API Pool

1 In the Connection Pool Manager, choose Add Connection Pool. The available user types display.

Fig. 7.3
Connection Pool Manager

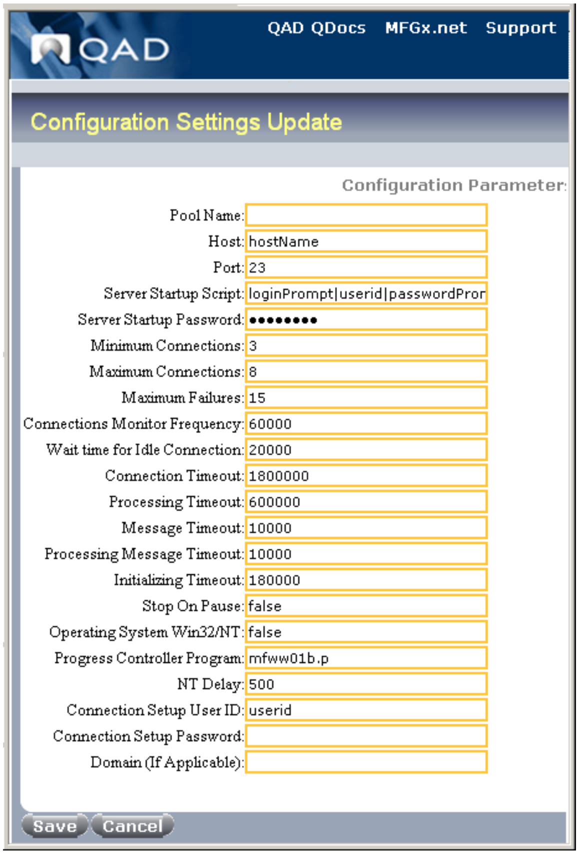


Connection Pool Manager

Status: Active

Status test-jits: Active
Status test-AP17: Active
Status JITS01: Active

2 Choose Add UIAPI Pool. The Configuration Settings Update form displays. Use the following field descriptions to create a connection pool configuration.



Configuration Settings Update

Configuration Parameter:

Pool Name:	
Host:	hostName
Port:	23
Server Startup Script:	loginPrompt|userid|passwordPror
Server Startup Password:	••••••••
Minimum Connections:	3
Maximum Connections:	8
Maximum Failures:	15
Connections Monitor Frequency:	60000
Wait time for Idle Connection:	20000
Connection Timeout:	1800000
Processing Timeout:	600000
Message Timeout:	10000
Processing Message Timeout:	10000
Initializing Timeout:	180000
Stop On Pause:	false
Operating System Win32/NT:	false
Progress Controller Program:	mfww01b.p
NT Delay:	500
Connection Setup User ID:	userid
Connection Setup Password:	
Domain (If Applicable):	

Fig. 7.4
Add a UI API Connection

Pool Name. The pool name must match the receiver name for QXtend Inbound API pools. The pool name displays in the view and delete connection pool menus.

Host. Enter the machine name or IP address of the telnet server.

Port. Enter the port number for the telnet server.

Server Startup Script. Enter the startup script for the telnet session. Specify the telnet server log-in prompts and the responses to these prompts separated with the pipe symbol (|). The standard order is: loginPrompt|userid|passwordPrompt|$PASSWD|osPrompt|startScript. For example:

See “Creating Telnet Log-In Scripts” on page 156.

```
login:|QXtend|Password:|password|$|exec ./qma.QXprod
```

Server Startup Password. Specify the password for the telnet session startup script (maximum 20 characters). The password is encrypted on entry. The startup script substitutes the encrypted password for the $PASSWD reference.

Minimum Connections. Enter the minimum number of open connections that the Connection Pool Manager should maintain. During startup, the Connection Pool Manager opens this number of connections. As connections are used, it continues to open more so that this number of open connections is maintained, until it reaches the value specified for Maximum Connections.

In general, keep this number as low as effectively possible; for example, 3 on faster systems. On slower systems, increase the number to reduce startup time on new requests.

Maximum Connections. Enter the maximum number of open connections that the system should allow. The Connection Pool Manager will not open any more connections than this.

Important On Windows systems, this field must be set to 2 or more to ensure successful connections.

Maximum Failures. Enter how many times the Connection Pool Manager should attempt to restart an unsuccessful connection. This number is reset when a successful connection is made. You can also reset it by using the Reset Failed Init Count command on the Connection Pool Functions menu.

Connections Monitor Frequency. Enter, in milliseconds, the interval for checking all connections. The default value is 180000 (3 minutes). This monitors all connections in all states and closes those that have timed out.

Wait Time for Idle Connection. When a connection is requested from the Connection Pool Manager, this timeout specifies the maximum wait for the connection. The maximum number of connections may have been reached, or new connections may be in the initializing state. The default value is 20000 (20 seconds).

Connection Timeout. Enter, in milliseconds, how long an HTML session can remain inactive before the Connection Pool Manager closes it. The default value is 1800000 (30 minutes).

Processing Timeout. Enter, in milliseconds, how long a connection can be in processing mode. Processing mode indicates a locked or busy screen. The default value is 3600000 (60 minutes). Connection Pool Manager closes locked or busy connections that exceed this.

Message Timeout. Enter the interval, in milliseconds, for Connection Pool Manager to wait for a general messaging reply from the telnet server. The default value is 10000 (10 seconds).

Processing Message Timeout. Enter the interval, in milliseconds, for Connection Pool Manager to wait for reply from the telnet server when a connection is in processing mode. The default value is 6666 (6.6 seconds).

Initializing Timeout. Enter the interval, in milliseconds, for Connection Pool Manager to wait for a telnet session to successfully initialize. The default value is 180000 (3 minutes).

Stop on Pause. For QXtend Inbound, this should always be set to false. This prevents a transaction from failing when a "Press Spacebar" message is displayed in the target MFG/PRO session.

Operating System Win32/NT. Set this to true if the Progress telnet sessions are executing on a computer with a Windows operating system. Otherwise, set this to false.

Progress Controller Program. Enter `mfww01b.p` for UI API pools.

NT Delay. This can safely be ignored for QXI connection pools.

Connection Setup User ID. This and the next two entries are the parameters required to connect to the target MFG/PRO instance. Enter the valid MFG/PRO user ID, such as `qxtend`.

Connection Setup Password. Enter the password for the MFG/PRO user ID (maximum 20 characters). The password is encrypted on entry.

Domain. Enter the valid MFG/PRO domain if the target instance has domains implemented. Domains were introduced in MFG/PRO eB2.1.

➧ See *User Guide: MFG/PRO eB2.1 New Features* for details on domains.

3 On completion, choose Save. The new connection pool is started automatically and is added to the list of connection pools in the Connection Pool Manager interface.

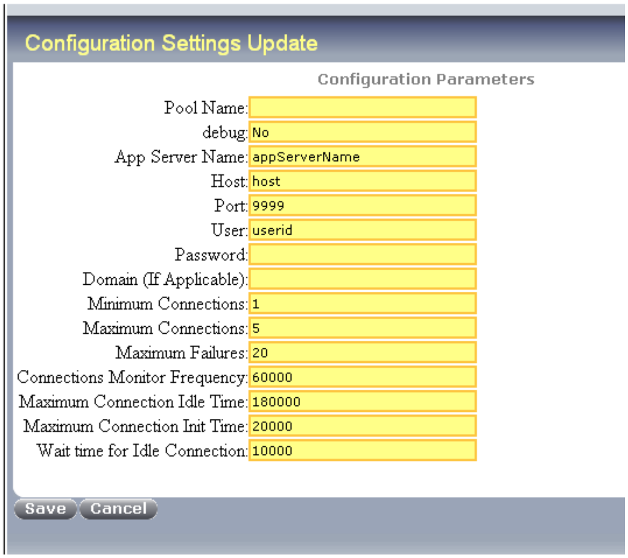
Adding a Code or JIT/S API Pool

The code and JIT/S API pools are slightly different than the XML-based pools for the UI API. You identify the Progress AppServer and the user ID and password for the AppServer. Several values covered for the UI API pools are not required for code API pools. You must also enter a session type for Progress Dynamics for JIT/S API pools.

1 In the Connection Pool Manager, choose Add Connection Pool. The available connection pool types display beneath the Add Connection Pool selection.

2 Choose Add Code API Pool or Add JIT/S API Pool as needed. The Configuration Settings Update form displays. Use the following field descriptions to create a connection pool configuration.

Fig. 7.5
Add a Code API Pool



Configuration Settings Update

Configuration Parameters

Pool Name:	
debug:	No
App Server Name:	appServerName
Host:	host
Port:	9999
User:	userid
Password:	
Domain (If Applicable):	
Minimum Connections:	1
Maximum Connections:	5
Maximum Failures:	20
Connections Monitor Frequency:	60000
Maximum Connection Idle Time:	180000
Maximum Connection Init Time:	20000
Wait time for Idle Connection:	10000

Pool Name. The pool name must match the receiver name for QXtend Inbound API pools. The pool name displays in the view and delete connection pool menus.

debug. If debug is set to Yes, debug information from the code adapter is written to the AppServer log file. If set to No, no AppServer logging occurs.

App Server Name. Enter the name given to the AppServer in the `ubroker.properties` file. This is configurable and is done by the user when creating the AppServer settings.

See "Progress AppServer Setup" on page 136.

Host. Enter the machine name or IP address of the machine where the Progress AppServer is installed and running.

Port. Enter the port number for the NameServer that is controlling the AppServer instance. The connection is requested from the NameServer and it assigns the AppServer to the connection.

User. Enter the MFG/PRO user name.

Password. This is the MFG/PRO user's password, encrypted on entry.

Domain. For a CodeAPI pool, enter the valid MFG/PRO domain if the target instance has domains implemented. Domains were introduced in MFG/PRO eB2.1. This field does not apply to JIT/S API pools.

See *User Guide: MFG/PRO eB2.1 New Features* for details.

Session Type. This value appears only in the JIT/S API connection pool configuration. Set it to dynamics session type to enable a Progress Dynamics session.

See *Installation Guide: QAD JIT Sequencing* for details.

Minimum Connections. Enter the minimum number of open connections that the Connection Pool Manager should maintain. During startup, the Connection Pool Manager opens this number of connections. As connections are used, it continues to open more so that this number of open connections is maintained, until it reaches the value specified for Maximum Connections.

In general, keep this number as low as effectively possible; for example, 3 on faster systems. On slower systems, increase the number to reduce startup time on new requests.

Maximum Connections. Enter the maximum number of open connections that the system should allow. The Connection Pool Manager will not open any more connections than this.

Maximum Failures. Enter how many times the Connection Pool Manager should attempt to start a connection. This number is reset when a successful connection is made. You can also reset it by using the Reset Failed Init Count command on the Connection Pool Function menu.

Connections Monitor Frequency. Enter, in milliseconds, the interval for checking all connections. The default value is 180000 (3 minutes). This monitors all connections in all states and closes those that have timed out.

Maximum Connection Idle Time. Enter, in milliseconds, the maximum time a 4GL connection can remain idle.

Maximum Connection Init Time. Enter, in milliseconds, the maximum time a 4GL connection can be initializing.

Wait Time for Idle Connection. Enter, in milliseconds, how long the Connection Pool Manager should wait for an initializing connection to become idle before an error is generated. The default value is 20000 (20 seconds).

3 On completion, choose Save. The new connection pool is started automatically and is added to the list of connection pools in the Connection Pool Manager interface.

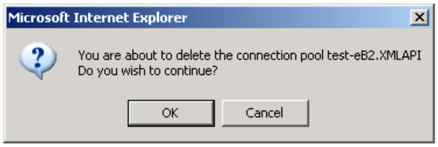
Delete a Connection Pool

When a connection pool is superseded by a new one, you can delete the older one.

1 In the Connection Pool Manager, choose Delete Connection Pool. The available connection pool types display beneath the Delete Connection Pool selection.

2 Select the pool to delete. A confirmation warning displays.

Fig. 7.6
Connection Pool Deletion Warning



Microsoft Internet Explorer

You are about to delete the connection pool test-eB2.XMLAPI
Do you wish to continue?

3 Choose OK to delete the connection pool.

Administering Connection Pools

Once connection pools have been created and started, the system administrator can access and manage the connection pools, individual connections within the pools, and user sessions. This section covers:

- Connection pool states
- Viewing connection pools
- Managing connection pools
- Managing user sessions

Connection Pool States

Each connection has one of the following statuses:

Initializing. A connection is just starting and is not yet available for use.

Idle. The connection is active and available for the next user request.

Busy. The connection is currently executing a user request.

Pause. The connection is waiting for a response from the user; for example, the user may need to press the spacebar to continue.

Processing. The connection is actively updating the Progress database; database records are locked.

Force Disconnect. This is a temporary state that occurs when the administrator closes an initializing connection.

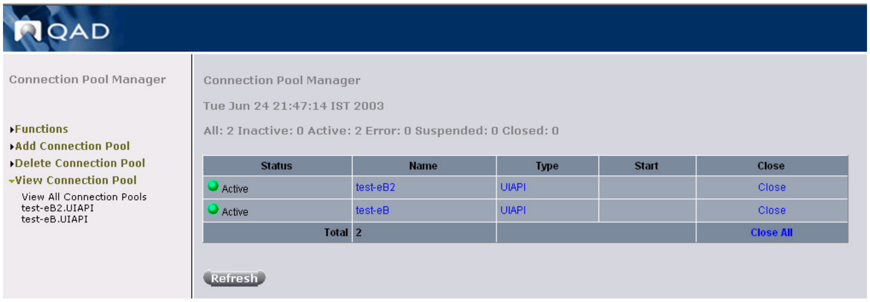
Disconnected. This is a temporary state that occurs when idle connections are closed.

Viewing Connection Pools

You can choose to view connections by status or view all connections.

1 In the Connection Pool Manager, choose View Connection Pool if it is not already expanded. The available connection pools display beneath View Connection Pool.

Fig. 7.7
View All Connection Pools



Connection Pool Manager

Tue Jun 24 21:47:14 IST 2003

All: 2 Inactive: 0 Active: 2 Error: 0 Suspended: 0 Closed: 0

Status	Name	Type	Start	Close
● Active	test-eB2	UIAPI		Close
● Active	test-eB	UIAPI		Close
Total	2			Close All

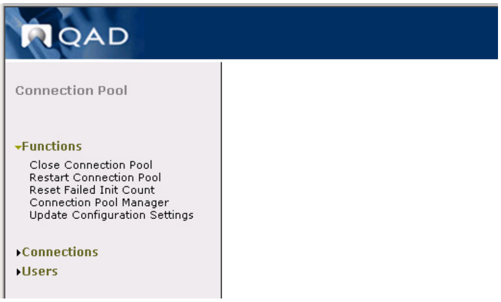
2 Choose an individual connection pool name from the View Connection Pool list to manage a specific pool.

3 For pools in the Processing state, a View link appears in the Close column. Use this feature when processing has stalled. Click on View to see the telnet session. Any error or wait-state message appears in the lower portion of the telnet screen.

Manage a Connection Pool

Once you select a specific connection pool, the menu options displayed are those specific to a connection pool.

Fig. 7.8
Connection Pool Menus



The following options are available from the Functions menu:

Close Connection Pool. This closes all connections in the current pool, ending all connections regardless of state or status. You can use Restart Connection Pool to start them up again.

Restart Connection Pool. This closes all connections in the current pool if they are open, and starts the minimum number of connections as defined in the configuration settings for the pool.

Reset Failed Init Count. The failed init (initialization) count is the number of times the server has attempted to launch a connection unsuccessfully. The number is automatically reset when a successful connection is made.

Connection Pool Manager. This closes the individual Connection Pool interface and returns you to the Connection Pool Manager.

Update Configuration Settings. This opens the configuration settings for the pool. The only difference is that the pool name is not available for editing.

Choose the Connections menu to view the pool of active connections. You can view the connections in the following categories:

See "Connection Pool States" on page 105 for information on the individual states.

All. Displays all connections. This is the only view that displays Disconnected and ForceDisconnect statuses.

Busy. Displays connections with Busy, Pause, or Processing status.

Idle. Displays all idle connections.

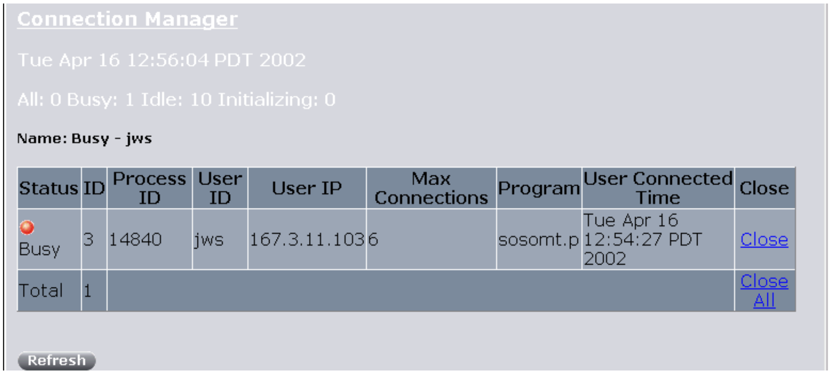
Initializing. Displays initializing connections.

Choose Refresh to update the screen display. Choose the Close link to close unneeded connections.

Manage User Sessions

Choose Users to view currently logged in users. Choose a user ID to see information related to that user. The users screen displays.

Fig. 7.9
Monitoring Users



Connection Manager

Tue Apr 16 12:56:04 PDT 2002

All: 0 Busy: 1 Idle: 10 Initializing: 0

Name: Busy - jws

Status	ID	Process ID	User ID	User IP	Max Connections	Program	User Connected Time	Close
● Busy	3	14840	jws	167.3.11.103	6	sosomt.p	Tue Apr 16 12:54:27 PDT 2002	Close
Total	1							Close All

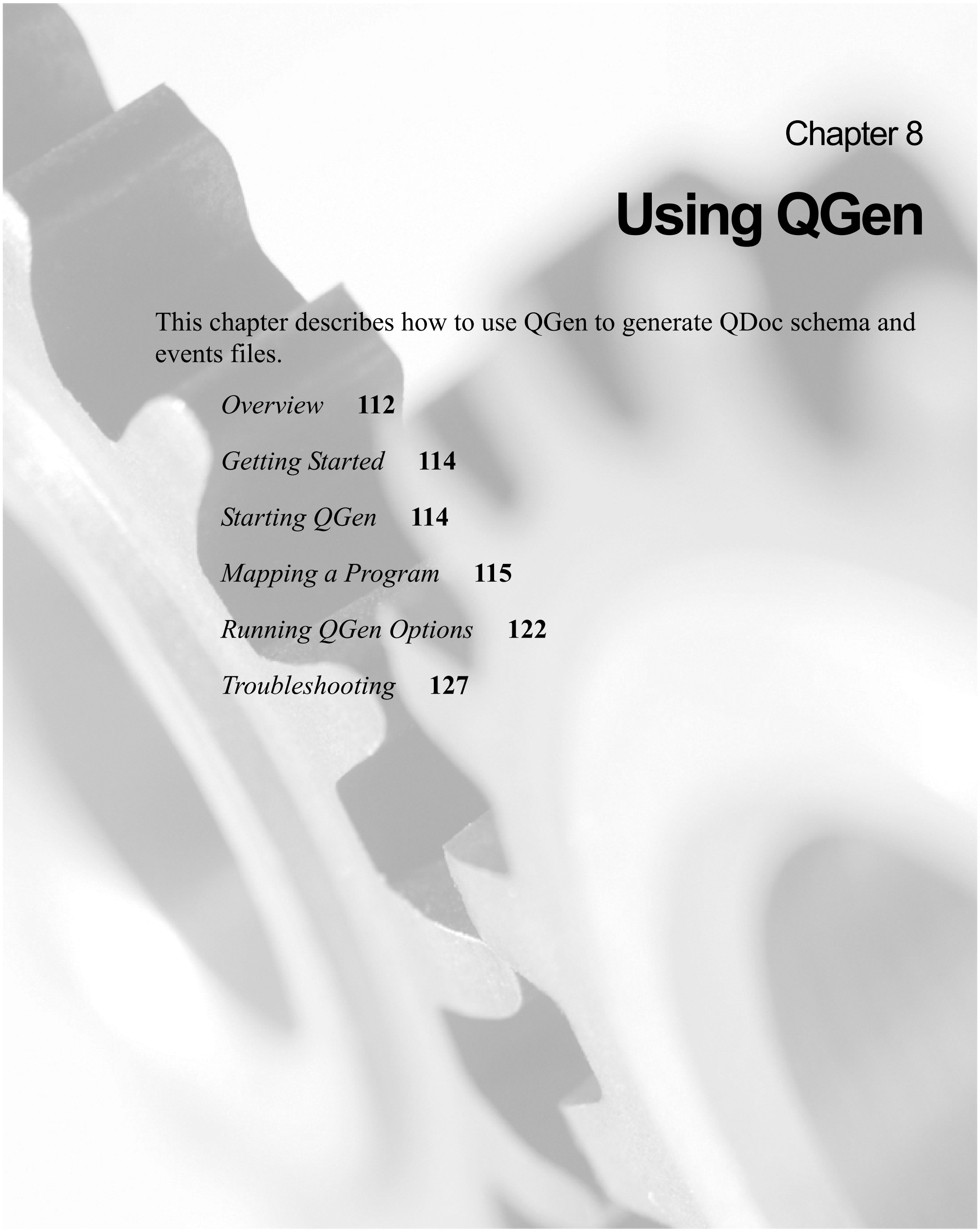
Choose Refresh to update the screen display. Choose the Close link to close a user session. For example, you can use this to closes a connection if a user has locked a database record and left their session running.

Note When QDocs are being processed, the user is the one defined as part of the server startup parameters in the Connection Manager Configuration Settings page. The IP address is that of the server machine where the QDocs were loaded into the queue for processing.

Section 3

Implementation

This section provides instructions for implementing optional features of QXtend Inbound in your environment.



Chapter 8

Using QGen

This chapter describes how to use QGen to generate QDoc schema and events files.

Overview

QGen runs in the background of specially-configured MFG/PRO character sessions to generate QDoc schema and events files for menu-level programs in MFG/PRO. The user initiates QGen from within the MFG/PRO session and logs field and iteration information. QGen stores this information as a master data file (`.dat`) for the calling procedure. A generate process converts these to the schema and events files.

See "Load" on page 123 for more information.

In general, QGen is intended for use in mapping custom programs or MFG/PRO programs that do not have released QDoc schemas and events files. You can also use QGen to modify the standard QAD QDoc schemas and events files by loading the standard `.dat` file and running through the menu-level program again.

The standard QGen process is:

- Launch an MFG/PRO QGen session.
- Map a program:
  - Open the MFG/PRO program.
  - Launch QGen.
  - Map individual fields and iterations.
  - Save your work.
- Generate the QDoc schemas and events file.

In addition, QGen enables you to:

- Modify individual field information after mapping is complete.
- Save partially mapped programs and reopen them later.
- Discard program maps and restart the mapping process.

Program Structure and Terminology

QGen follows the MFG/PRO interface to identify fields and navigation options. For the purposes of creating a QGen data (`.dat`) file, the different QGen options are based on the type of field you are currently mapping.

As you proceed through an MFG/PRO program, you encounter the following types of fields:

Primary Key. The definition of the primary key is different depending on whether you are building a schema for use with the code API or with the UI API. If the QDoc is processed using the code API, the primary key defines the relationship between parent and child records; if not correctly specified, this relationship is not populated when obtaining the data from the request XML. As a result, some processing may fail.

In this case, a primary key is a field that MFG/PRO uses to locate or create the unique records required during this pass through the program. For Sales Order Maintenance, for example, this is the sales order number and the sales order line numbers. If you are not sure about the primary key for a program, it is usually the active field or fields available when you first enter the program or iteration frame; this is not necessarily the fields in the first frame.

If the QDoc is processed using the UI API, the primary key is used only to determine which fields are populated in the response document; only the fields defined as primary keys appear in the response schema.

Delete Field. A delete field is any field, usually all grouped in a single frame, from which you can delete the current record. These fields are easily identified: the status line at the bottom of the MFG/PRO screen displays F5=Delete as an available function key.

First Entry Event. A first entry event is one that controls which branch of a program you descend. Usually, these fields have default values and are not accessed directly during an update; instead they are accessed by pressing F4. For example, the single/multiple line option in Sales Order Maintenance is accessed by pressing F4 from the sales order line field.

Choose Event. A choose event is a field in a selection list.

Iteration. An iteration is a repetition of a grouping of data, such as lines on a sales order. For QGen, an iteration is identified as a field that QGen returns to for a second pass-through. For sales order lines, this is the sales order line number. When QGen returns to this field the second time, it displays an iteration frame where you can name the iteration.

Getting Started

For more information, see “Install QXtend Inbound Files” on page 31.

The QGen tool is installed as part of the QXtend installation. When the installation is complete, the following files are installed in `/mfgsvr/qxtendserver/tools`:

- `eventsGenerator.p`
- `fileMaint.i`
- `generateDocs.p`
- `getFrameInfo.p`
- `programMapper.p`
- `runProgramMapper.p`
- `schemaGenerator.p`

In addition, a `.dat` file is installed in `/mfgsvr/qxtendserver/tools/datFiles`:

- `commonTypes.dat`

Starting QGen

To start QGen, you create a startup script for an MFG/PRO character client that calls QGen. The following instructions start with a copy of an existing MFG/PRO startup script:

1. Open an MFG/PRO startup script such as `clientProduction.sh` and save it as the QGen startup script with a name such as `clientQGen.sh`.

2. Modify the PROPATH from:

```
SET PROPATH=.,/mfgsvr,/mfgsvr/bbi,
```

To:

```
SET PROPATH=/mfgsvr/qxtendserver,.,/mfgsvr,/mfgsvr/bbi,
```

3. In the last line modify `mf.p` to `tools/runProgramMapper.p` and add `mfgwrapper=true,apimode=true,mnemonics_raw=true` to the `-param` value:

   ```
   exec $DLC/bin/_progres -param mfgwrapper=true,apimode=true,
   mnemonics_raw=true -p tools/runProgramMapper.p
   -pf $MFG_HOME/Production.pf -d mdy
   ```

4. Save the revised script.

5. Run the revised startup script.

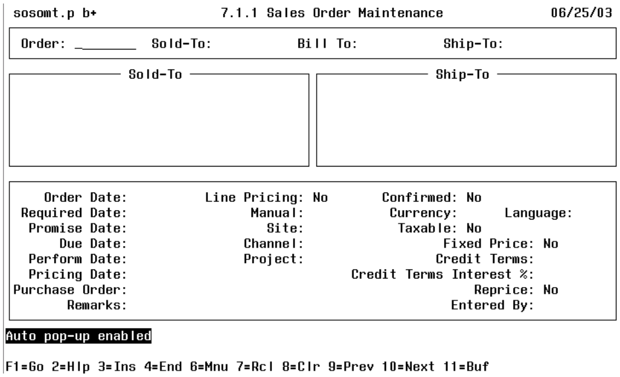
This starts an MFG/PRO session with the QGen tool running in the background.

Important Use this MFG/PRO session only to generate QDocs, not for entering data.

Mapping a Program

1. Navigate to the program in MFG/PRO that you want to generate QDocs for.

2. In the first field, press Ctrl+W. A message, Auto pop-up enabled, displays.



```
 sosomt.p b+               7.1.1 Sales Order Maintenance                06/25/03
  Order: ________  Sold-To:          Bill To:          Ship-To:
                Sold-To                              Ship-To

     Order Date:          Line Pricing: No       Confirmed: No
  Required Date:                Manual:           Currency:      Language:
   Promise Date:                  Site:            Taxable: No
       Due Date:               Channel:                   Fixed Price: No
   Perform Date:               Project:                  Credit Terms:
   Pricing Date:                              Credit Terms Interest %:
 Purchase Order:                                              Reprice: No
        Remarks:                                           Entered By:
Auto pop-up enabled

F1=Go 2=Hlp 3=Ins 4=End 6=Mnu 7=Rcl 8=Clr 9=Prev 10=Next 11=Buf
```

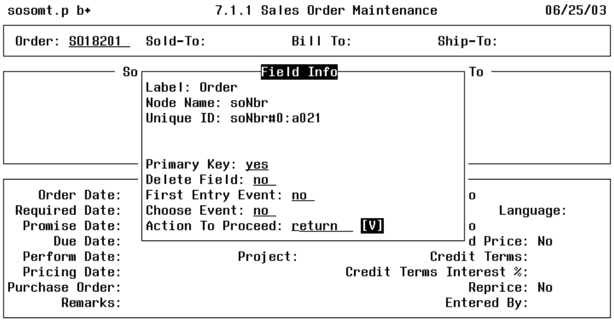
Fig. 8.1
Pressing Ctrl+W in a QGen Session

3. Enter data in the first field or select an existing record to edit and press Enter to move to the next field.

Important You must press Enter on each field in the program to launch the automatic pop-up for each field. If you press Go to move from a field, you may move to the next frame and the remaining fields in the previous frame are not mapped.

The Field Info pop-up appears. For each field, enter the required values.

Fig. 8.2
Field Info Pop-Up



```
sosomt.p b+                7.1.1 Sales Order Maintenance            06/25/03
 Order: SO18201    Sold-To:           Bill To:           Ship-To:
               So               Field Info              To
                  Label: Order
                  Node Name: soNbr
                  Unique ID: soNbr#0:a021


                  Primary Key: yes
                  Delete Field: no
    Order Date:   First Entry Event: no                 o
 Required Date:   Choose Event: no                          Language:
  Promise Date:   Action To Proceed: return   [V]       o
      Due Date:                                         d Price: No
  Perform Date:             Project:               Credit Terms:
  Pricing Date:                         Credit Terms Interest %:
Purchase Order:                                         Reprice: No
       Remarks:                                      Entered By:
```

Label. The label value is the field label that displays on the MFG/PRO screen. This is not available for editing.

Node Name. This is the field name displayed in humpback notation to conform to XML standards. For example, so_nbr is displayed as soNbr. This is not available for editing. Underscores are removed from the node names.

Unique ID. This is an alphanumeric value generated during mapping of a field. It is used to distinguish between different fields in a program that use the same node name. This is not available for editing.

For more on iterations, see "Map Iterations" on page 119.

Primary Key. This field has different uses in the code and UI APIs. For code APIs, the primary key defines the relationship between parent and child. A Yes in this field designates this field as one of the primary keys of the parent or child records.

For UI APIs, set this to Yes on any field that you want to include in the response schema.

Delete Field. If you can delete a record when the cursor is in this field—for example if the F5=Delete option appears in the program status line—set this to Yes. Set this to Yes for all fields in a frame that allow a deletion. Otherwise, accept the default of No.

First Entry Event. Set this field to Yes if a special action is required the first time the field is encountered in an incoming QDoc request. These are usually fields that have navigation consequences, such as the Line Format (S/M) field in Sales Order Maintenance (7.1.1) that controls whether the sales order lines will be entered line-by-line or in multi-line mode.

Choose Event. The default value is No. Change this to Yes if the field is a field in a selection list.

Action to proceed. Press the down arrow to open the drop-down list on this field and select the key required to leave the current field and move to the next field.

4 Press Go to save and exit the pop-up data. "Updated: <*field name*>" displays at the bottom of the screen.

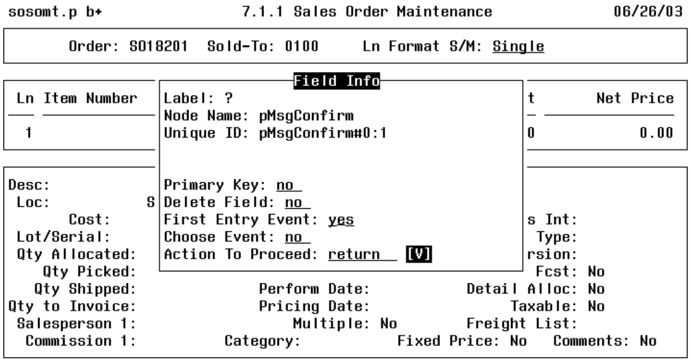
5 If First Entry Event for the field is set to Yes, the First Entry Event pop-up displays. Otherwise, you move to the next field.

6 If you move to the next field, enter data and press Enter. The Field Info pop-up displays for this field. By default, the key you used to exit the field is used for the Action to Proceed value.

Map First Entry Events

A first entry event occurs at a branching point in a program, usually prior to starting an iteration. Typically the field is accessed by pressing F4 from the first field of an iteration.

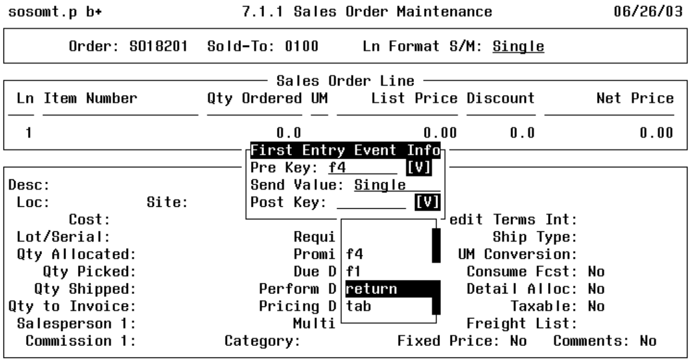
Fig. 8.3
First Entry Event
Field Definition



```
sosomt.p b+              7.1.1 Sales Order Maintenance             06/26/03
+--------------------------------------------------------------------------+
|       Order: SO18201  Sold-To: 0100     Ln Format S/M: Single            |
+--------------------------------------------------------------------------+
+-----------------+-------------Field Info--------------+------------------+
| Ln Item Number  |Label: ?                             |t      Net Price  |
| -- ------------ |Node Name: pMsgConfirm               |-  -------------- |
|  1              |Unique ID: pMsgConfirm#0:1           |0          0.00   |
+-----------------|                                     |------------------+
|Desc:            |                                     |                  |
|                 |Primary Key: no                      |                  |
| Loc:           S|Delete Field: no                     |                  |
|       Cost:     |First Entry Event: yes               |s Int:            |
| Lot/Serial:     |Choose Event: no                     | Type:            |
| Qty Allocated:  |Action To Proceed: return    [V]     |rsion:            |
|   Qty Picked:   +-------------------------------------+ Fcst: No         |
|  Qty Shipped:             Perform Date:          Detail Alloc: No        |
|Qty to Invoice:            Pricing Date:              Taxable: No         |
| Salesperson 1:                Multiple: No       Freight List:           |
|  Commission 1:          Category:          Fixed Price: No   Comments: No|
+--------------------------------------------------------------------------+
```

1 When you leave a first entry event field, the Field Info pop-up displays. Enter Yes for First Entry Event and press Go. The First Entry Event Info pop-up displays.

Fig. 8.4
First Entry Event
Info Pop-Up



```
sosomt.p b+              7.1.1 Sales Order Maintenance             06/26/03
+--------------------------------------------------------------------------+
|       Order: SO18201  Sold-To: 0100     Ln Format S/M: Single            |
+--------------------------------------------------------------------------+
+---------------------------- Sales Order Line ----------------------------+
| Ln Item Number      Qty Ordered UM    List Price Discount      Net Price |
| -- --------------   ----------- --  ---------- --------  -------------- |
|  1                          0.0           0.00      0.0           0.00 |
+----------------------------+First Entry Event Info+----------------------+
                             |Pre Key: f4       [V] |
+----------------------------|Send Value: Single    |----------------------+
|Desc:                       |Post Key:         [V] |                      |
| Loc:           Site:       +-----------+          +                      |
|       Cost:                            |          |edit Terms Int:       |
| Lot/Serial:                       Requi|          |    Ship Type:        |
| Qty Allocated:                    Promi|f4        | UM Conversion:       |
|   Qty Picked:                     Due D|f1        |  Consume Fcst: No    |
|  Qty Shipped:                 Perform D|return    |  Detail Alloc: No    |
|Qty to Invoice:                Pricing D|tab       |       Taxable: No    |
| Salesperson 1:                    Multi+----------+  Freight List:       |
|  Commission 1:          Category:          Fixed Price: No   Comments: No|
+--------------------------------------------------------------------------+
```

2 Use the down arrow to open the available keystroke information. Press Enter to select a value, then press F4 to close the drop-down menu. Use the following information to enter field data:

Pre Key. By default, the F4 key appears because this is the typical key to access First Entry Event fields. Press the down arrow to open the drop-down list on this field and select a different key to access the First Entry Event field if necessary.

Send Value. Specify the value that must be entered in the First Entry Event field to follow the data entry path you want to use. For example, to set Line Format (S/M) in Sales Order Maintenance to Multi, enter `m` or `multi` in Send Value.

Post Key. This is the key required to leave the current field and move to the next field. The default is Return (the Enter key). Press the down arrow to open the drop-down list on this field and select a different key to exit the First Entry Event field if necessary.

3 Press Go to save and exit the pop-up.

Map Iterations

When you come to a frame that repeats—lines in a sales order, items in a container, and so on—QGen recognizes this as an iteration. This means the QGen mapper maps the first field in the iteration twice.

To accomplish this, you navigate through the first field of the frame twice. The first time, enter data normally as described in “Mapping a Program” on page 115. The second time through, make sure to return to the first field in the frame. QGen automatically recognizes the iteration.

One common iteration type is the Comments iteration. This does not need to be mapped because it is defined in the `commonTypes.dat` file.

➧ See “Getting Started” on page 114.

At a Comments field, press Enter twice. The first Enter pops up the Field Info screen and captures the field information. The second detects the iteration. Enter `transComments` as the Iteration Name. Exit the Comments iteration by pressing F4 until you reach the next field, and continue through the remaining fields of the program.

False Iterations

You will also encounter false iterations in MFG/PRO. These are fields that are entered into more than once, but that are not receiving new data. For example, when you leave the Consignment Location field in Sales Order Maintenance, several messages that require user input may display regarding the status of inventory in this location. After responding to the first message, the interface returns you to the field. QGen sees this as an iteration. In these instances, press F4 to leave the Iteration Info pop-up and continue with the next field in the program.

Important Each menu-level program is itself an iteration. When you return to the first field in a program—the sales order number field in Sales Order Maintenance—select the field again to allow QGen to detect the iteration. If you skip this step, an error occurs during QDoc generation.

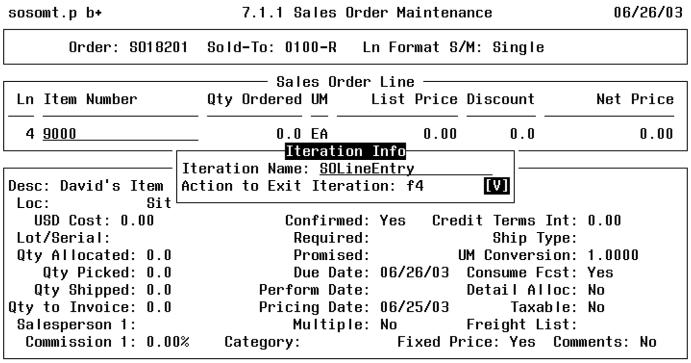
Hidden Iterations

In general, iterations are detected automatically by QGen. In some cases, such as when multiple values are allowed in a single frame, the iteration is not detected because you have not left the frame. Adding multiple users per domain in User Group Maintenance (36.3.4) is one example of this. In this case, you must open the Field Info pop-up on the first instance of the field to map the field. When you arrive back at the field after entering data, you must press Ctrl-F again to open the Iteration Info pop-up.

Mapping an Iteration

In most cases when you enter an iterating field the second time, the Iteration Info pop-up displays automatically.

Fig. 8.5
Iteration Info Pop-Up



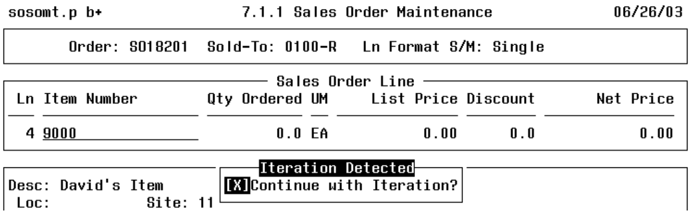
```
sosomt.p b+               7.1.1 Sales Order Maintenance               06/26/03

         Order: SO18201  Sold-To: 0100-R   Ln Format S/M: Single

                               Sales Order Line
 Ln Item Number            Qty Ordered UM     List Price Discount       Net Price

  4 9000                           0.0 EA           0.00       0.0           0.00
                                 Iteration Info
                        Iteration Name: SOLineEntry
Desc: David's Item      Action to Exit Iteration: f4          [V]
 Loc:              Sit
   USD Cost: 0.00                     Confirmed: Yes   Credit Terms Int: 0.00
 Lot/Serial:                           Required:              Ship Type:
 Qty Allocated: 0.0                    Promised:          UM Conversion: 1.0000
    Qty Picked: 0.0                    Due Date: 06/26/03  Consume Fcst: Yes
   Qty Shipped: 0.0                Perform Date:           Detail Alloc: No
Qty to Invoice: 0.0                Pricing Date: 06/25/03       Taxable: No
 Salesperson 1:                        Multiple: No        Freight List:
  Commission 1: 0.00%    Category:             Fixed Price: Yes  Comments: No
```

Enter the following data:

Iteration Name. Enter a name for the iteration. This iteration name is the name that will appear in the QDoc. It must conform to the standard humpback notation for QDocs; for example, maintainSalesOrder.

Action to Exit Iteration. Enter the key required to exit the iteration (usually F4). Use the down arrow to select an alternative value if necessary.

4 Press Go to exit the pop-up. Continue with Iteration? appears.



```
sosomt.p b+                  7.1.1 Sales Order Maintenance                06/26/03
      Order: SO18201   Sold-To: 0100-R   Ln Format S/M: Single
                          Sales Order Line
 Ln Item Number          Qty Ordered UM     List Price Discount         Net Price
  4 9000                         0.0 EA           0.00       0.0             0.00
                        Iteration Detected
Desc: David's Item      [X]Continue with Iteration?
 Loc:             Site: 11
```

Fig. 8.6
Continue With Iteration Query

5 From the Iteration Detected pop-up, there are two courses of action:

a Press spacebar to enter an X and press Go to continue through the iteration again. You can do this as many times as there are unique pathways in the iteration. This moves you to the second field of the iteration and you would continue through the fields until all iterations are mapped.

Note Fields that were mapped on the first pass through the program display the message "<*fieldname*> found in table" in the program message area. The Field Info pop-up does not display.

b Press Go without the selecting the continue option to exit the iteration. A message displays:

```
Please exit the iteration.
```

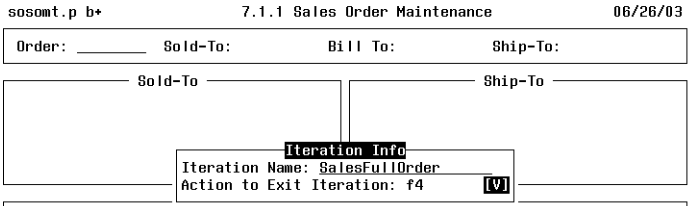
Press spacebar to select the message. Press Go to exit the iteration message. Move on to step 6.

6 Press the appropriate key, usually F4, to exit the first iteration field.

Note If the iteration is exited in a different way than you originally specified (Action to Exit Iteration was set to F4 but should be F4:F4), you can change it in Update Mode.

See "Change Mode to Update" on page 124.

When you return to the first field in the program, remember to press Enter, as though you were going to go through the program again. The Iteration Info pop-up displays. Enter a name for the full program iteration.



```
sosomt.p b+                  7.1.1 Sales Order Maintenance                06/26/03
 Order: ________    Sold-To:             Bill To:             Ship-To:
                     Sold-To                          Ship-To
                             Iteration Info
                  Iteration Name: SalesFullOrder
                  Action to Exit Iteration: f4            [V]
```

Fig. 8.7
Complete Program Iteration

To finish the mapping process, press Ctrl+W to disable the pop-ups.

Running QGen Options

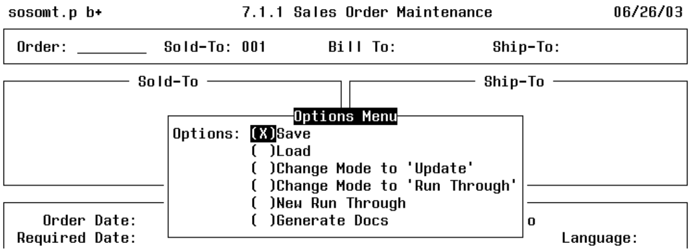
At any time during an MFG/PRO QGen session, press Ctrl+O to open the QGen Options menu. The options are:

- Save
- Load
- Change Mode to Update
- Change Mode to Run Through
- New Run Through
- Generate Docs

Save

This option saves the information that has been mapped during the session. You can save incomplete program run-throughs as well as completed ones.

Fig. 8.8
QGen Options
Menu on Save

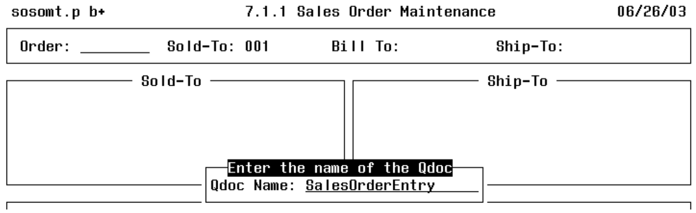


```
sosomt.p b+               7.1.1 Sales Order Maintenance              06/26/03
 Order: ________   Sold-To: 001        Bill To:             Ship-To:
                   Sold-To                                Ship-To
                                Options Menu
                      Options: (X)Save
                               ( )Load
                               ( )Change Mode to 'Update'
                               ( )Change Mode to 'Run Through'
                               ( )New Run Through
                               ( )Generate Docs                 o
    Order Date:
 Required Date:                                                     Language:
```

1 Press spacebar to select Save if it is not already selected.

2 Press Go to save.

3 You are asked to enter the name of the QDoc. Use humpback notation; for example, SalesOrderEntry.



```
sosomt.p b+                 7.1.1 Sales Order Maintenance                06/26/03
 Order: ________   Sold-To: 001        Bill To:             Ship-To:
                 Sold-To                                 Ship-To
                             Enter the name of the Qdoc
                             Qdoc Name: SalesOrderEntry
```

Fig. 8.9
QDoc Save Frame

4 Press Go to save the QDoc information. The message “Finished saving data” appears. The QDoc information is stored in a `.dat` file, for example `SalesOrderEntry.dat`, in the `/mfgsvr/qxtendserver/tools/datFiles` directory.

Load

To continue a previously incomplete mapping session, load a saved `.dat` file.

1 Arrow to Load and press spacebar.

2 Press Go to load the `.dat` file.

3 You are prompted to enter the name of the file; for example, SalesOrderEntry. The QDoc must exist in the correct location and the capitalization must be correct. The `.dat` extension is not required.

4 Press Go to load the document into the current QGen session. The message “Finished loading data” appears.

5 Open the MFG/PRO program the `.dat` file was created from.

6 Tab through the program fields. Each mapped field displays the message:

```
<fieldname> found in table.
```

The Field Info pop-up displays automatically at the first unmapped field.

See “Mapping a Program” on page 115.

7 Select the Save option to save any changes (Ctrl+O, Save).

Change Mode to Update

Use Update mode when you want to modify attributes of fields you have already mapped. This set of steps assumes you have a `.dat` file already loaded. In Update mode, the field order of the MFG/PRO menu-level program is ignored.

1. Arrow to Change Mode to Update and press spacebar.
2. Press Go to switch to Update mode.
3. Open the MFG/PRO program associated with the data map you want to update.
4. Tab through the program fields. When you reach a field you want to update, press Ctrl+F. The Field Info pop-up displays with the existing field information.
5. Modify the information and press Go to close and save the changes for the field.
6. After completing your updates, select the Save option to save any changes to the `.dat` file.

Change Mode to Run Through

Run Through allows new field information to be mapped. Use this option to change back from Update mode.

1. Arrow to Change Mode to Run Through and press spacebar.
2. Press Go to switch to Run Through mode.
3. Map programs as described in “Mapping a Program” on page 115.
4. Select the Save option to save your work.

New Run Through

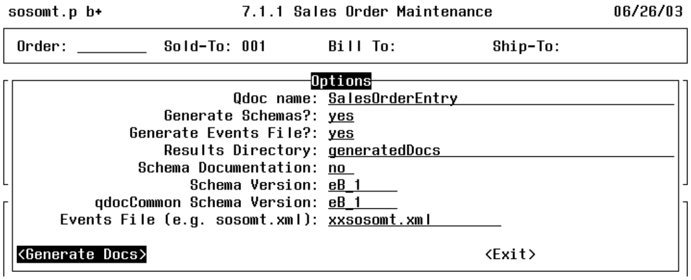
New Run Through deletes the existing session information and restarts the mapping process.

1. Arrow to New Run Through and press spacebar.
2. Press Go to start a new run through.
3. You are asked, “Are you sure you wish to reset?” Press spacebar to confirm and press Go. The session information is deleted.
4. Open the MFG/PRO program you want to remap and press Ctrl+W to start the mapping process.

Generate Docs

This option generates a QDoc schema and events file from the program information that has been mapped and saved in a `.dat` file.

1. Arrow to Generate Docs and press spacebar.
2. Press Go to start the generation. The Generate Options frame displays.



```
sosomt.p b+                  7.1.1 Sales Order Maintenance                 06/26/03

 Order: ________   Sold-To: 001        Bill To:             Ship-To:

                                    Options
                          Qdoc name: SalesOrderEntry
                  Generate Schemas?: yes
              Generate Events File?: yes
                  Results Directory: generatedDocs
               Schema Documentation: no
                     Schema Version: eB_1
            qdocCommon Schema Version: eB_1
        Events File (e.g. sosomt.xml): xxsosomt.xml

 <Generate Docs>                                       <Exit>
```

Fig. 8.10
Generate QDocs Pop-up

3. Fill in the information using the field descriptions below:

   QDoc name. Enter the QDoc name as saved in the QGen `.dat` file; for example, `SalesOrderEntry`.

   Generate Schemas. Set to Yes to generate schemas.

   Generate Events File. Set to Yes to generate an events file.

Results Directory. Specify the target directory where the schema and events files are created. This defaults to `generatedDocs`. The directory is created automatically during successful document creation if it does not already exist. For example:

```
/mfgsvr/qxtendserver/tools/generatedDocs
```

Schema Documentation. Set to Yes, the MFG/PRO screen label for each field is added to the schema file and a directory called `documentedSchemas` is created below the `generatedDocs` directory where the newly created schema files are placed.

If No, MFG/PRO screen labels are ignored, and a directory called `undocumentedSchemas` is created below the `generatedDocs` directory where the newly created schema file is placed.

Schema Version. The default value is eB_1. Change it to the required version; for example, eB2_3 for the third version of the file in the eB2 release.

qdocCommon Schema Version. The default value is eB_1. Modify it to the required version; for example, eB2_3 for the third version of the file in the eB2 release.

Events File. This should be made up of the MFG/PRO program name—for example, `sosomt` (Sales Order Maintenance)—plus an `.xml` extension: `sosomt.xml`. The events file is created in the `generatedDocs` directory.

To create additional events files for the same MFG/PRO program name, enter a file name with some version information you can easily track such as `sosomt_01.xml`.

4. Choose Generate Docs. The message "Finished generating documents" appears. The files are created in the `generatedDocs` directory.

5. Press Enter to close the "Finished" message. You return to the Generate Options screen. Tab to Exit and press Enter to return to MFG/PRO.

6. Place your generated schema and events files in the correct Tomcat directories before use. For standard and custom schema files, follow the instructions in "Add a Schema to the Master Lists" on page 87. Specify the events file path during the addition of the schema.

Troubleshooting

This section documents some possible error conditions that can occur while using QGen and offers suggested resolutions.

Problem: The Automatic pop-up is enabled, but a pop-up is not displayed on exiting the field.

Solution 1: First ensure that the mode is Run Through and not Update.

Solution 2: If QGen is in Run Through mode, check if you are in the message area. This is the area at the bottom of the screen where messages pop up—the cursor is at a question requiring a Yes/No answer. If so, the Field Info pop-up does not display. Continue past the field in the message area. When the next field is reached, QGen displays a pop-up containing the field information for the field in the message area before displaying a pop-up containing information for the currently selected field.

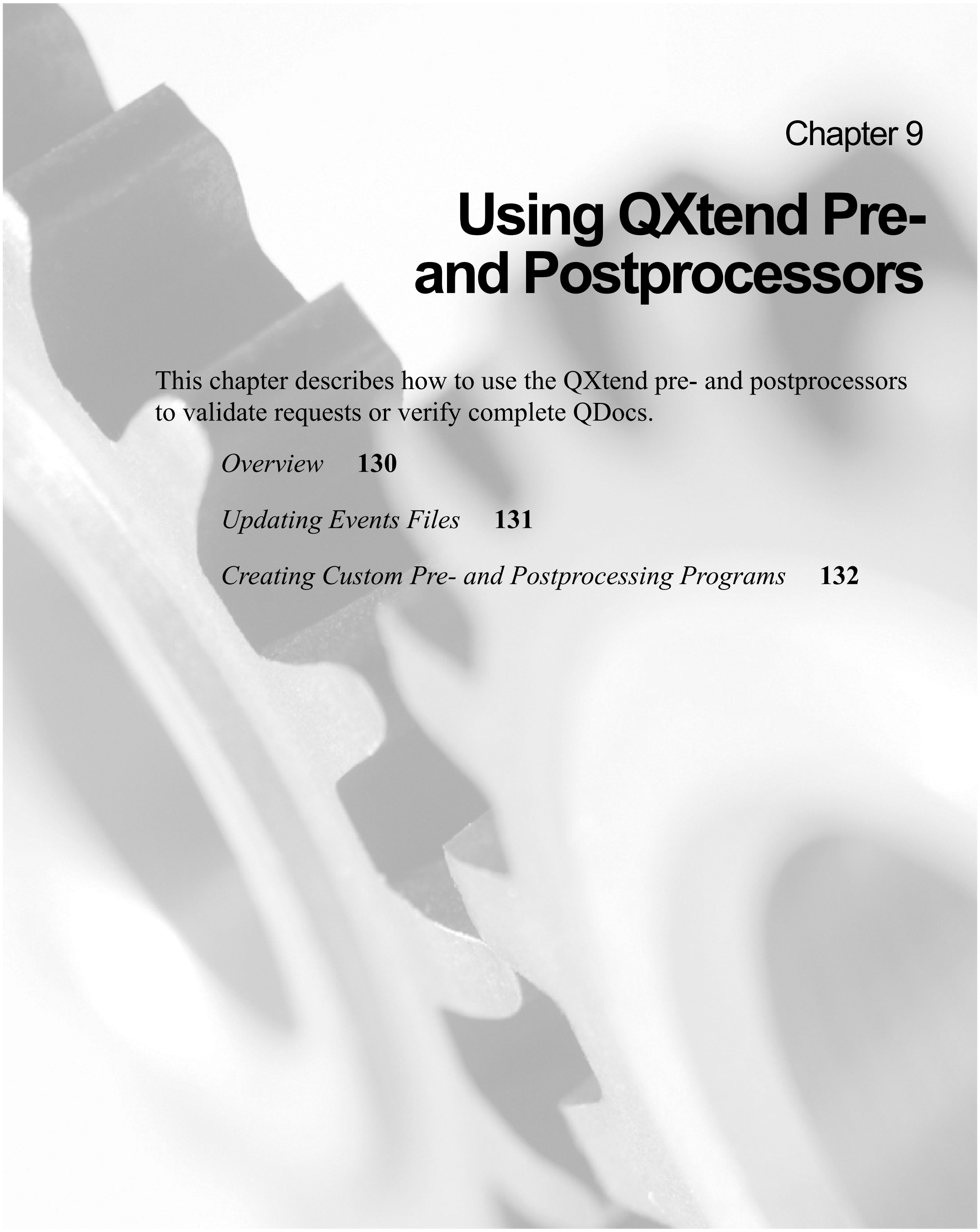
Solution 3: If you are not in the message area, disable the automatic pop-up using Ctrl+W, then tab back to the field and use Ctrl+F to manually map the field.

Error message: Unable to create directory nnnn

This error occurs when the current user does not have permissions to create the specified directory.

Error message: The file “qdoc.dat” has no root node name

The first field in a program was not mapped as an iteration. When mapping a program, the first field in the program must be selected again and an iteration name entered. Exit the Generate Docs menu. Tab to the first field in the program and press Ctrl+F to map it.



Chapter 9

Using QXtend Pre- and Postprocessors

This chapter describes how to use the QXtend pre- and postprocessors to validate requests or verify complete QDocs.

Overview

The QXtend preprocessor lets you call custom programs to validate or modify inbound QDocs prior to writing data to MFG/PRO. The preprocessor programs can eliminate MFG/PRO errors, filter required records, eliminate records based on unique sets of criteria, or add or remove data as required.

The postprocessor is similar but is used to validate and modify the response document contents. One example is obtaining updated data from MFG/PRO, such as line item prices that changed after the initial data update from the QDoc request.

The method for implementing and using the QXtend pre- and postprocessors is:

- Enable pre- and postprocessing capability.
- Update the events files to reference the pre- or postprocessing program.
- Create the custom pre- or postprocessor program.

Enabling Pre- and Postprocessing in QXtend

See “Include MFG/PRO Field Values” on page 50.

To initiate the ability to run pre- and postprocessing in QXtend, edit the `qxtendconfig.xml` file. If you have configured QXtend to enable field values in response documents, you should have already made this change.

1 Open *TOMCAT_HOME*`/webapps/qxtendserver/WEB-INF/conf/qxtendconfig.xml`.

2 Locate the `messageServletURL` attribute within the `<general-config>` node.

3 Enter the correct host and port values to communicate with the QXtend server.

```
<messageServletURL label=“Message Servlet URL” value=“http:
//<host>:<port>/qxtendserver/MessageReceiverServlet”/>
```

4 Save and close the file.

Updating Events Files

The events files in QXtend Inbound are `.xml` files containing program-specific keystroke and navigation. Events files for each QDoc-supported program are shipped with the product or are generated by the QGen product. You can maintain multiple copies of the events files for each MFG/PRO program to support either different paths through a program, or to test incremental performance improvements in update paths.

These files must first be copied to the custom events directory; then a new event and the new program name must be added.

1 Copy the events file you want to modify to the `custom` directory. The following example is for sales orders. Copy from:

```
TOMCAT_HOME/webapps/qxtendserver/WEB-INF/events/eB2/
sosomt.xml
```

To:

```
TOMCAT_HOME/webapps/qxtendserver/WEB-INF/events/eB2/
custom/sosomt.xml
```

2 Edit the events schema file to add a new IterationEvent. The event must have an attribute of either `preprocess` or `postprocess`. This attribute defines the custom program to call when the iteration event is encountered. Use the following code fragment showing a preprocessing program as a model:

```
<field uid="soNbr#0:a02" name="so_nbr" inheritevents="true">
    <IterationEvent iterationname="salesOrder" exititeration=
    "f4" preprocess="xxMyPreProg.p"/>

</field>
```

This sample runs `xxMyPreProg.p` before processing the QDoc.

An equivalent postprocessing program call would appear as follows:

```
<field uid="soNbr#0:a02" name="so_nbr" inheritevents="true">
    <IterationEvent iterationname="salesOrder" exititeration=
    "f4" postprocess="xxMyPostProg.p"/>

</field>
```

Creating Custom Pre- and Postprocessing Programs

The custom pre- and postprocessing programs must be written in the Progress 4GL. The program you create is called from a Progress program named `mfww01b.p`. This Progress program has procedures that your program can call to limit your programming task to the validations you require.

Program Structure

The program structure is as follows:

```
DEFINE INPUT PARAMETER pQDoc AS HANDLE NO-UNDO.
DEFINE INPUT PARAMETER pMessageLogger AS HANDLE NO-UNDO.
    /* Custom validation code goes here; */
    /* Log any warnings or errors.*/
RETURN ERROR.
```

Program Naming and Location

Store the file in the standard custom code directories defined for MFG/PRO:

MFGPROInstallDir`/us/xx/xxsosomt.p`

This path should be included in your PROPATH, and should reflect the actual installation language if it is not US. The standard QAD naming convention for custom programs is to prefix the file name with `xx`. You may need to adjust the file name if customized programs already exist for the program—for example, change `xxsosomt.p` to `xxsosoqx.p` if `xxsosomt.p` already exists.

QDoc Iterations

Iterations are steps through the data nodes, such as sales order lines, of a given inbound QDoc. These nodes and their contents are passed into the Progress calling program as an X-DOCUMENT. The input parameter `pQDoc` is a handle to the Progress X-DOCUMENT. The custom program can access the nodes within this X-DOCUMENT and carry out any validation required.

Note The custom program is not given access to all of the QDoc, only the current iteration node.

For example, a preprocessing program for SOLine iterates on each sales order line. In the sample QDoc below, the bold salesOrderLineDetail iteration is the current iteration when the event triggers for sales order line 1. When the event triggers for sales order line 2, the italic salesOrderLineDetail iteration is the current iteration.

```
<maintainSalesOrder xmlns="http://www.qad.com/qdoc/eb"
xmlns:enc="http://www.w3.org/2002/12/soap-encoding" xmlns:qcom=
"http://www.qad.com/qdoc/common" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance" xsi:schemaLocation="http://www.qad.com/
qdoc/eb ..\..\..\schemas\eB2\maintainSalesOrder-eB2_1.xsd"
version="eB2_1" scopeTransaction="false" suppressResponseDetail=
"true" xml:lang="en">
    <salesOrder>
        <soNbr>dq10009</soNbr>
        <soCust>1012000</soCust>
        <soConsignment>no</soConsignment>
        <salesOrderLineDetail>
            <line>1</line>
            <sodPart>ALX001</sodPart>
            <sodtaxable1>no</sodtaxable1>
            <sodcmmts>no</sodcmmts>
        </salesOrderLineDetail>
        <salesOrderLineDetail>
            <line>2</line>
            <sodPart>ALX002</sodPart>
            <sodtaxable1>no</sodtaxable1>
            <sodcmmts>no</sodcmmts>
        </salesOrderLineDetail>
    </salesOrder>
</maintainSalesOrder>
```

Warnings and Errors

If an error occurs, the custom program must log the error using the logError procedure in `mfww01b.p`, then return `Error` to stop processing the QDoc. If an `Error` is returned, the next QDoc is selected and preprocessing is run on it. If there is a warning or the processing is successful, your program should return nothing.

You can log all warnings and errors that arise during pre- or postprocessing by calling the `logWarning` or `logError` subprocedures in `mfww01b.p`.

To log a warning message:

```
run logWarning in pMessageLogger (input "Warning: sales order
SO9945 not confirmed.").
```

To log an error message:

```
run logError in pMessageLogger (input "Error: sales order 9956
incorrect so_nbr format.").
```

Any messages logged using these methods appear in the response QDoc.

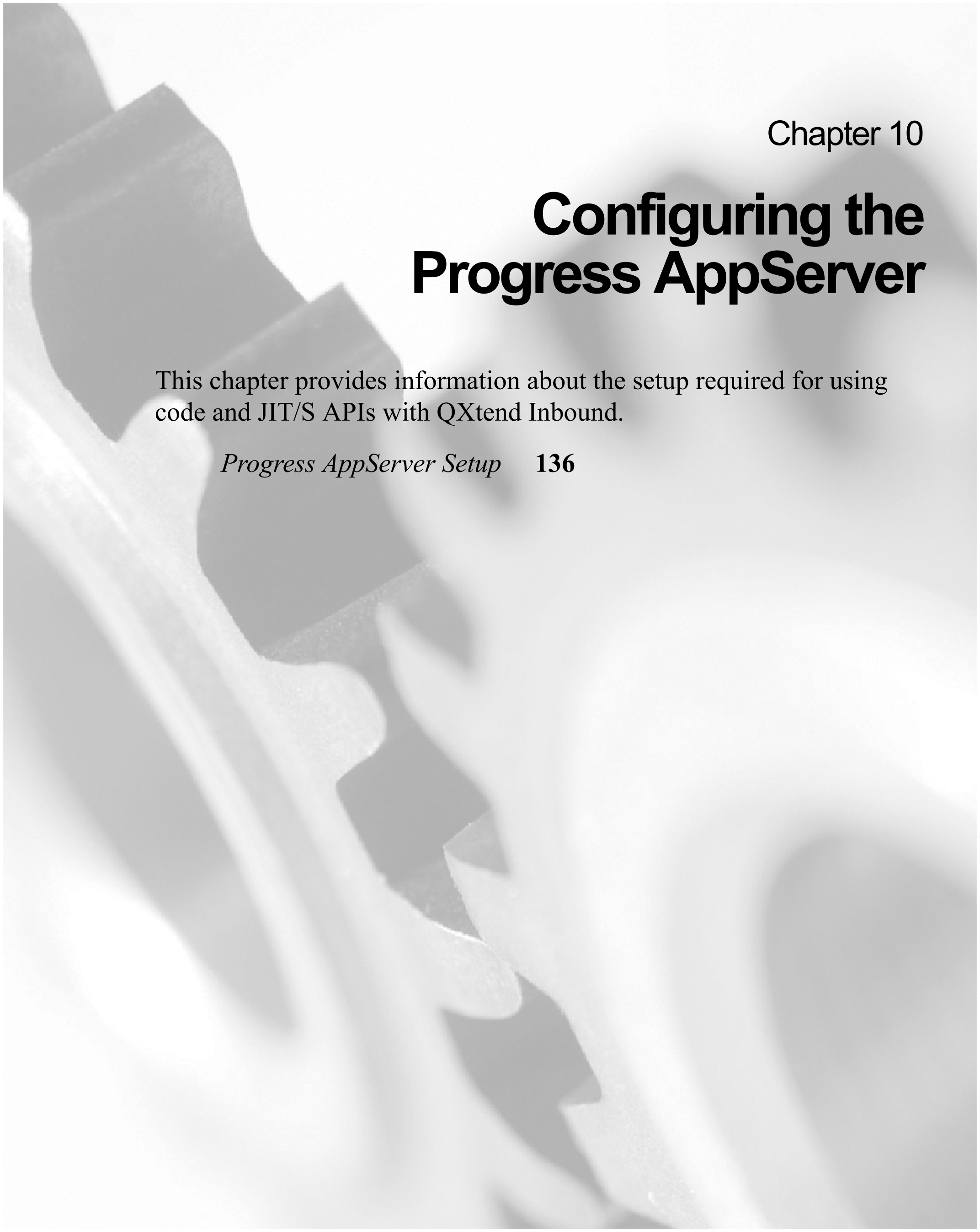
Note Logging an error does not stop the QDoc from processing; you must return `Error` to stop it from processing.

Modify the QDoc Iteration Node

If your custom program modifies the QDoc iteration node that is passed down, the program must run the `updateXMLDocument` procedure provided in `mfww01b.p`. The `updateXMLDocument` procedure requires a handle to the new updated X-DOCUMENT as a parameter. Here is an example of how you would call the `updateXMLDocument` procedure:

```
run updateXMLDocument in pMessageLogger (pQDoc).
```

Calling `updateXMLDocument` as illustrated here ensures the iteration node that was sent to the preprocess program is updated with `pQDoc`.



Chapter 10

Configuring the Progress AppServer

This chapter provides information about the setup required for using code and JIT/S APIs with QXtend Inbound.

Progress AppServer Setup

QXtend Inbound relies on the Progress open client and AppServer for the available code and JIT/S APIs in order to process synchronous API requests. The code and JIT/S APIs shipping with QXtend Inbound are:

- maintainSupplier (`advnai01.p`)
- maintainPurchaseOrder (`popoai01.p`) for both discrete and scheduled orders
- maintainSupplierSchedule (`rssai01.p`)
- receiptsBackwardExplode (`icunrc01.p`)

Windows Setup

If the AppServers are running on Windows, use the Progress Explorer tool to configure the NameServer and the AppServer instances. See "Progress NameServer Setup" on page 137 and "Progress AppServer Setup" on page 138 for details on the configuration of the NameServer and AppServer instance properties.

The AdminServer, NameServer, and AppServer can be started and stopped using the Progress Explorer tool and they can also be set to start automatically. All administration for the AppServer and the AdminServer can be done using the Progress Explorer tool.

Non-Windows OS Setup

Configure the AppServer manually by editing scripts in the Progress install directory. The information in this section does not detail every possible setup scenario. For more information on AdminServer, NameServer, and AppServer configuration, see the Progress documentation.

Progress AdminServer Details

The Progress AdminServer is started using the `proadsv` script. The AdminServer uses the `<DLC>/AdminServerPlugins.dat` file to start the AdminServer by default. Default ports used for the AdminServer are:

- AdminServer Port: 20931

- AdminServer to Database Communication Port: 7834

These ports are the Progress defaults and can be set by passing startup parameters.

See “Parameter File Setup” on page 137.

The port the AdminServer is running on is used when starting and stopping NameServer and AppServer connections. Consideration must be given to which ports are going to be used for implementation if the default ports are not available or acceptable.

Progress NameServer Setup

The NameServers are configured in the `ubroker.properties` file. This file resides in the *DLC*`/properties` directory and contains all the configuration settings for all the NameServers and AppServers.

The following example shows how to set up a NameServer instance. A port must be allocated to each NameServer instance:

```
[NameServer.NS1]
    srvrLogFile=<Log DIR>/NS1.ns.log
    environment=NS1
    autoStart=1
    portNumber=4091
    host=<qxtendhost>
```

Set up the properties in the `ubroker.properties` file for the NameServer that will control the QXtend Inbound AppServers.

Verify NameServer settings by running:

```
$DLC/bin/nsconfig -f <property file> -i <NameServer>
```

Validate NameServer settings by running:

```
$DLC/bin/nsconfig -f <property file> -i <NameServer> -v
```

Parameter File Setup

The AppServer connects to the MFG/PRO databases based on values defined in a parameter file. This parameter file is similar to those used when starting a normal MFG/PRO session with some minor differences.

The parameter file can contain any of the client startup parameters that are required. It should not contain a `-p` (startup procedure) because this is controlled by the AppServer settings. The following is an example of a parameter file that is connecting to remote database servers:

```
-db mfgprod -ld qaddb -H hp40 -S mfg-server -N TCP -trig triggers
-db admprod -ld qadadm -H hp40 -S adm-server -N TCP
-db helpprod -ld qadhelp -H hp40 -S help-server -N TCP
-d mdy -yy 1920 -Bt 350 -c 30 -D 100 -mmax 3000 -nb 200 -s 63
```

The following is an example of a parameter file that is connecting to local database servers:

```
-db /mfgpro/db/mfgprod -ld qaddb -znotrim -trig triggers
-db  /mfgpro/db/hlpprod -ld qadhelp
-db  /mfgpro/db/admprod -ld qadadm
-d mdy -yy 1920 -Bt 350 -c 30 -D 100 -mmax 3000 -nb 200 -s 63
```

Progress AppServer Setup

The AppServers are also configured in the `ubroker.properties` file. When creating a new AppServer outside of the Progress Explorer, a uuid (universal unique identifier) must be specified. The ID must be unique because the controlling NameServer uses it to identify the broker. To generate a uuid, use the following command:

```
$DLC/bin/genuuid
```

Copy the generated uuid into the `ubroker.properties` file.

The following sample shows how an AppServer definition in the `ubroker.properties` file would be configured for QXtend Inbound. Bold type indicates settings that may require changes for the QXtend Inbound AppServer setup.

```
#
# Sample properties for AppServer hosting MFG/PRO APIs.
# Most of these entries are shown for illustrative purposes only.
#
[UBroker.AS.mfgproapi]
#
#operatingMode should be set to Stateless for the MFG/PRO APIs.
#
operatingMode=Stateless
classMain=com.progress.ubroker.broker.ubroker
srvrExecFile=${DLC}/bin/_proapsv
srvrMinPort=2002
srvrMaxPort=2202
defaultService=0
debuggerEnabled=0
4glSrcCompile=0
description=AppServer Broker
srvrLogFile=/qad/web/server/logs/mfgproapi.server.log
   brokerLogFile=/qad/web/server/logs/mfgproapi.broker.log
   portNumber=52779
   initialSrvrInstance=1
   maxSrvrInstance=5
   autoTrimTimeout=300
   appserviceNameList=mfgproapi
   controllingNameServer=NS1
   environment=mfgproapi
   uuid=167.3.8.223:1dacd46a:eabdb5a8d4:-8000
   description=AppServer for MFG/PRO API
   srvrStartupParam=-pf mfgproapi.pf
   PROPATH=/mfgsvr/qxtendserver/xrc:/mfgsvr:/mfgsvr/us/bbi:.
#
# Link MFG/PRO API programs to AppServer sessions.
# These entries are required.
#
   srvrStartupProc=mfaistrt.p
   srvrShutdownProc=mfaishut.p
   srvrConnectProc=mfaicon.p
   srvrDisconnProc=mfaidis.p
   srvrActivateProc=mfaiact.p
   srvrDeactivateProc=mfaidea.p
```

The `controlling NameServer` entry should point to the NameServer that was defined in the previous section.

See “Progress NameServer Setup” on page 137.

Modify the PROPATH

Add the QXtend Inbound r-code directory to the PROPATH; for example:

```
/mfgsvr/qxtendserver/xrc
```

Warning If the AppServer PROPATH is changed and the AdminServer is running, stopping and starting the AppServer is not sufficient to pick up the new PROPATH. You must restart the AppServer, NameServer, and AdminServer.

Important QXtend AppServers set up for JIT/S must have `*JITSInstallDir*/jpssrc/apiint/app` at the beginning of the PROPATH.

Verify the Implementation

Verify the settings for the AppServer by running the following command and checking the entries:

```
$DLC/bin/asconfig -f <property file> -i <NameServer>
```

Validate the AppServer settings by running:

```
$DLC/bin/asconfig -f <property file> -i <NameServer> -v
```

Commands to Start Servers

Start the AdminServer on default ports:

```
proadsv -start
```

Start the AdminServer and specify the ports to use:

```
proadsv -start -port <AdminServer Port> -adminport <AdminServer
to Database Port>
```

Start the NameServer:

```
nsman -port <AdminServer Port> -i <NameServer Name> -x
```

Start the AppServer:

```
asbman -port <AdminServer Port> -i <AppServer Name> -x
```

Commands to Query Servers

Query the NameServer:

```
nsman -port <AdminServer Port> -i <NameServer Name> -q
```

Query the AppServer:

```
asbman -port <AdminServer Port> -i <AppServer Name> -q
```

Commands to Stop Servers

Stop the AdminServer:

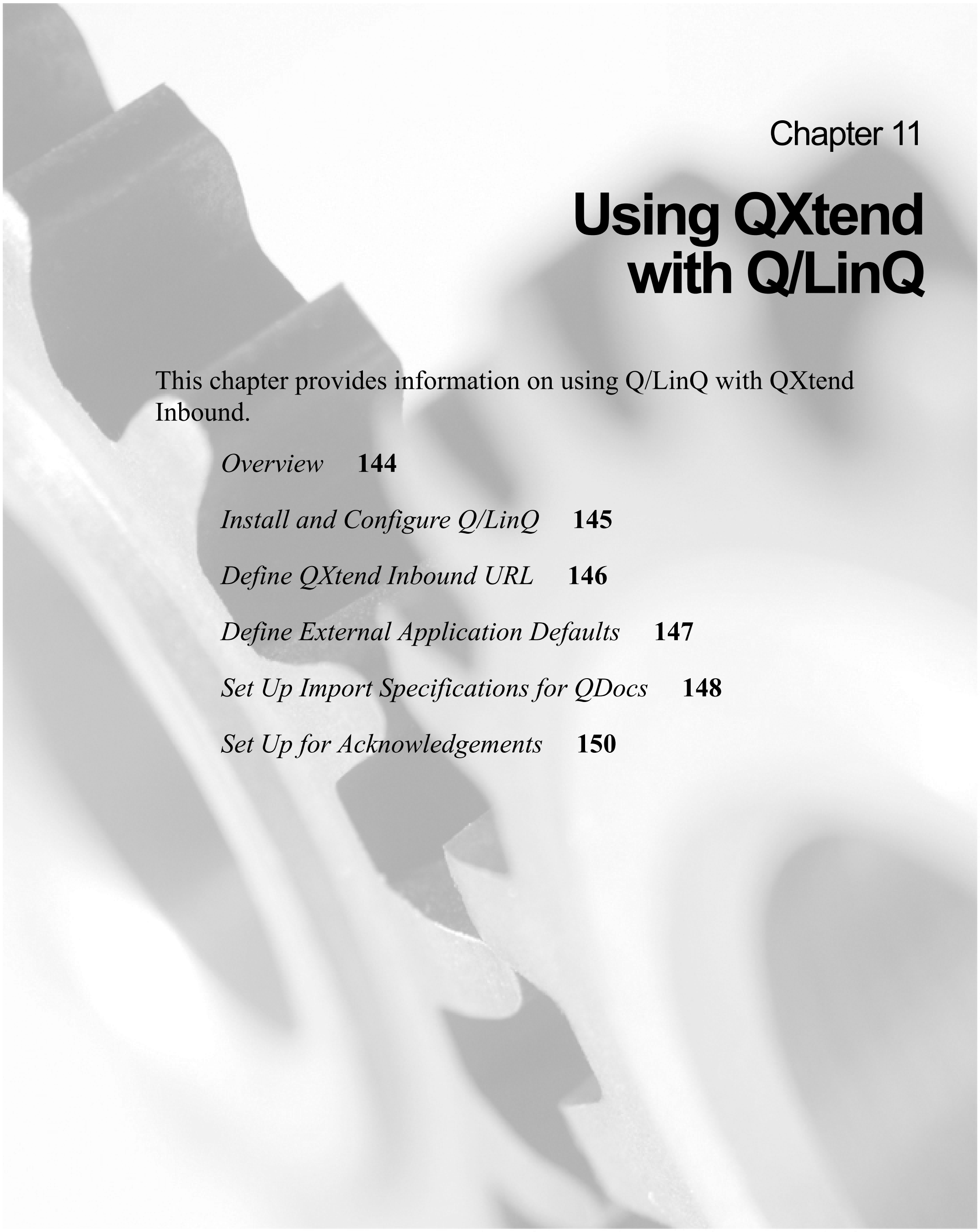
```
proadsv -stop
```

Stop the NameServer:

```
nsman -port <AdminServer Port> -i <NameServer Name> -e
```

Stop the AppServer:

```
asbman -port <AdminServer Port> -i <AppServer Name> -e
```



Chapter 11

Using QXtend with Q/LinQ

This chapter provides information on using Q/LinQ with QXtend Inbound.

Overview

See *External Interface Guide: Q/LinQ* for details about installing and using Q/LinQ.

Q/LinQ is a tool set for building integrations with third-party or in-house applications for complex data exchange. It also provides infrastructure for administering and managing the data exchange between MFG/PRO and these external applications.

Q/LinQ and QXtend Inbound provide overlapping features in terms of queuing messages to be processed as MFG/PRO transactions. However, the processing of data into MFG/PRO through QDocs is a more robust method than the more CIM-like user-interface-emulation approach generally used by earlier releases of Q/LinQ. If you have installed and configured Q/LinQ and now install QXtend Inbound, you can use the services of QXtend to process messages received and managed by Q/LinQ through its processing API.

There are a number of reasons why you might want to use Q/LinQ and QXtend Inbound together:

1. Q/LinQ does outbound processing, while QXtend is currently only available for inbound documents.
2. Q/LinQ has APIs to perform data mapping.
3. Q/LinQ has middleware communications capability and the associated feature of generating and processing confirmations.
4. Q/LinQ stores in-process documents in a database rather than externally as files. This approach provides enhanced data security and scalability and reduces the overall number of pieces that must be managed in the integration.

In addition, you can continue to process some Q/LinQ import documents using UI emulation, while redirecting processing responsibility to QXtend for others.

To process QDocs, you need to follow some special steps when you set up and configure Q/LinQ:

1 Install Q/LinQ based on standard installation instructions, giving special consideration to the setting in the `QqMomAdapter.ini` file that affects processing of QDocs.

2 Specify the location of the QXtend Inbound server in Q/LinQ Control (36.8.24).

3 Define appropriate defaults in Register External Application (36.8.1.1).

4 Define import specifications for QDocs using Import Specification Maintenance (36.8.1.3).

5 Define import and export specifications for the acknowledgement documents passed between MFG/PRO and QXtend Inbound.

Install and Configure Q/LinQ

You should install and configure Q/LinQ using the instructions in *External Interface Guide: Q/LinQ* in conjunction with the information related to the most current service pack for Q/LinQ. To use Q/LinQ with QXtend, you must have one of the following minimum release levels:

- MFG/PRO eB Service Pack 7 or higher
- MFG/PRO eB2 Service Pack 4 or higher
- MFG/PRO eB2.1, any version

A section in the Q/LinQ MOM adapter initialization file (`QqMomAdapter.ini`) supports the use of Q/LinQ with QXtend Inbound. The `[envelope usage]` section indicates for each supported Q/LinQ external application ID whether it uses a QDoc envelope for inbound and outbound communications. Setting this parameter to True indicates to Q/LinQ that it should remove the SOAP envelope before processing a document received from a specific application ID.

Table 11.1 lists the variable in the `[envelope usage]` section of the Q/LinQ MOM Adapter.

Table 11.1
Envelope Section

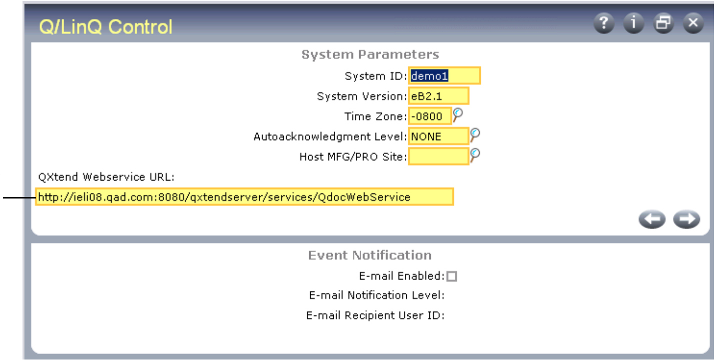
Variable	Description
<application ID value>	True if all messages received from and sent to the external application will use the QDoc envelope; otherwise, false. If omitted, false is assumed.

Setting the value to false or omitting it causes the Q/LinQ MOM adapter to behave as it did before the introduction of QDocs.

Define QXtend Inbound URL

Specify the URL that identifies the location of QXtend in your system (maximum 70 characters) in Q/LinQ Control (36.8.24).

Fig. 11.1
Q/LinQ Control (36.8.24), QXtend Webservice URL



Q/LinQ Control

System Parameters

System ID: demo1
System Version: eB2.1
Time Zone: -0800
Autoacknowledgment Level: NONE
Host MFG/PRO Site:
QXtend Webservice URL:
http://ieli08.qad.com:8080/qxtendserver/services/QdocWebService

Event Notification

E-mail Enabled: □
E-mail Notification Level:
E-mail Recipient User ID:

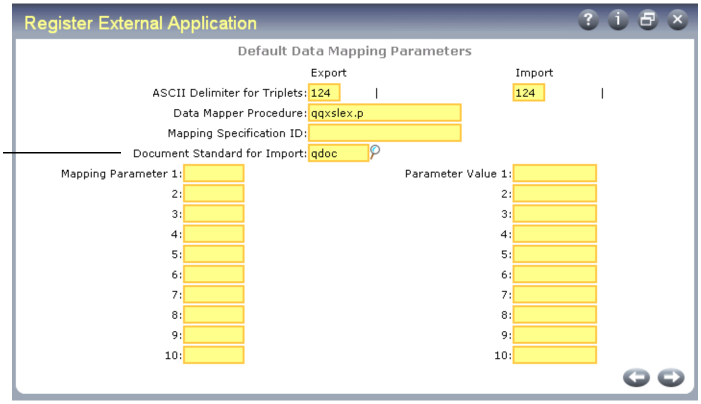
Update this field to use Q/LinQ with QXtend Inbound.

Q/LinQ uses this URL to locate the QXtend server when the import specification associated with a document indicates that it should be processed through QXtend.

Define External Application Defaults

In Register External Application (36.8.1.1), enter the unique IDs you have decided upon for identifying specific connections. Settings in two frames affect the relationship between Q/LinQ and QXtend.

In the Default Data Mapping Parameters frame, review the setting for Document Standard for Import.



Register External Application

Default Data Mapping Parameters

	Export	Import
ASCII Delimiter for Triplets:	124 |	124 |
Data Mapper Procedure:	qqxslex.p	
Mapping Specification ID:		
Document Standard for Import:	qdoc	

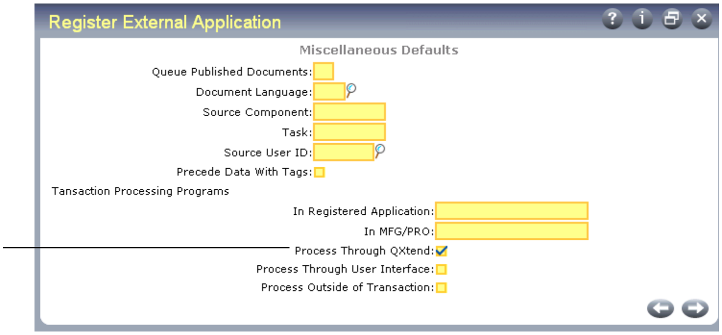
Mapping Parameter 1:		Parameter Value 1:	
2:		2:	
3:		3:	
4:		4:	
5:		5:	
6:		6:	
7:		7:	
8:		8:	
9:		9:	
10:		10:	

Set to qdoc if most messages from this ID are processed through QXtend.

Fig. 11.2 Register External Application (36.8.1.1), Default Data Mapping Parameters

If most of the documents you are going to process using this connection will be mapped to QDocs, specify qdoc in this field. Q/LinQ uses this field to determine how to process a message when a specific import specification cannot be found.

In the Miscellaneous Defaults frame, set the appropriate value for Process Through QXtend.



Register External Application

Miscellaneous Defaults

Queue Published Documents:
Document Language:
Source Component:
Task:
Source User ID:
Precede Data With Tags: □
Tansaction Processing Programs
In Registered Application:
In MFG/PRO:
Process Through QXtend: ☑
Process Through User Interface: □
Process Outside of Transaction: □

Set this field to Yes to process QDocs.

Fig. 11.3 Register External Application, Miscellaneous Defaults

The value you set here defaults to the same field in Import Specification Maintenance when you define a new import specification for this application. If you plan to process most of the documents from this external application as QDocs, set this field to Yes.

Set Up Import Specifications for QDocs

Use Import Specification Maintenance (36.8.1.3) to set document-specific parameter values for importing messages to be processed as QDocs.

Many fields in this program default from the values you specify in Register External Application.

Matching Specifications to Documents

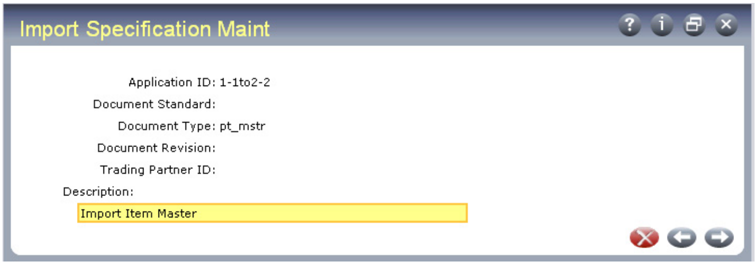
You can use up to five values to define an import specification: document standard, document type, document revision, application ID, and trading partner ID. The only required field is the document type. The system uses the following logic to find a specification to apply to a document:

1. It looks for one with an exact match for document standard, document type, document revision, application ID, and trading partner ID.
2. It looks for one with matching document standard, document type, document revision, application ID, and a blank trading partner ID.
3. It looks for one with matching document standard, document type, document revision, and blank application and trading partner IDs.

This lets you set up generic specifications that can apply to all documents of a certain type (and optional standard and revision) regardless of the particular application or trading partner associated with a document.

Register Import Specifications

The key field in Import Specification Maintenance (36.8.1.3) related to QXtend is Process Through QXtend.



Import Specification Maint

Application ID: 1-1to2-2
Document Standard:
Document Type: pt_mstr
Document Revision:
Trading Partner ID:
Description:
Import Item Master

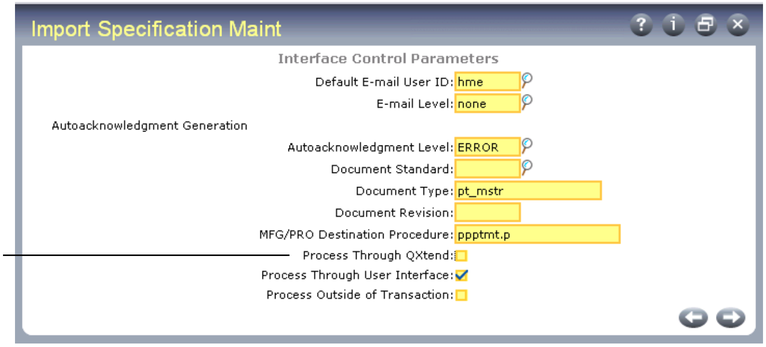
Fig. 11.4 Import Specification Maintenance (36.8.1.3)

Application ID. Enter the name of an external application. Leave blank if you want this specification to apply to all documents of a certain type, standard, and revision without regard to the associated application.

Document Standard. Enter qdoc as the document standard to process documents through QXtend.

Document Type. Enter a user-defined name for the type of data that is being processed.

Document Revision and Trading Partner ID can be left blank. If they are used, Document Standard, Document Type, Document Revision, and Trading Partner ID must be a unique combination of values.



Import Specification Maint

Interface Control Parameters
Default E-mail User ID: hme
E-mail Level: none
Autoacknowledgment Generation
Autoacknowledgment Level: ERROR
Document Standard:
Document Type: pt_mstr
Document Revision:
MFG/PRO Destination Procedure: ppptmt.p
Process Through QXtend: □
Process Through User Interface: ☑
Process Outside of Transaction: □

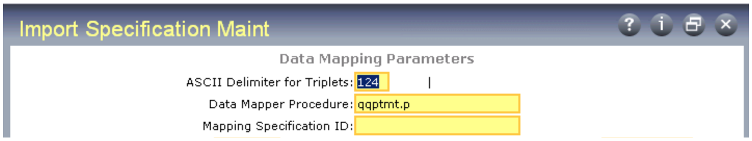
Set this field to Yes to process QDocs.

Fig. 11.5 Import Specification Maintenance, Interface Control Parameters

In the Interface Control Parameters frame, set up incoming processing parameters.

> *MFG/PRO Destination Procedure.* Leave this field blank to process the incoming document through QXtend. When Process Through QXtend is Yes, the system automatically invokes `qqqdocpr.p` to process the imported document as a QDoc.
>
> *Process Through QXtend.* Indicate if you want the documents associated with this import specification to be processed using QXtend Inbound.
>
> No: Documents are processed through user interface emulation or gateway programs.
>
> Yes: Documents are processed as QDocs through QXtend Inbound.
>
> ***Note*** When this field is Yes, the document is treated as a QDoc regardless of the setting for Process Through User Interface.
>
> *Process Through User Interface.* Enter Yes to invoke the destination procedure through the CIM Interface. When Process Through QXtend is Yes, this field is assumed to be No.

Fig. 11.6 Import Specification Maintenance, Data Mapping Parameters



Data Mapping Parameters

ASCII Delimiter for Triplets: 124

Data Mapper Procedure: qqptmt.p

Mapping Specification ID:

In the Data Mapping Parameters frame, enter the user-written mapping program that maps the input data to the QDoc XML format.

Set Up for Acknowledgements

When you use Q/LinQ and QXtend together, QDoc confirmation documents can be sent between the two applications to acknowledge the receipt and processing of messages. Specific mapping procedures and XSLT stylesheets must be used to map these documents to the format required by the receiving application.

Use the values in Table 11.2 in the Data Mapping Parameters frame of Export Specification Maintenance (36.8.1.2) to map QDoc confirmations to the CONFIRM_BOD format used by Q/LinQ.

Table 11.2 Mapping Values for Exporting Acknowledgements

BOD:	`CONFIRM_BOD`
XSLT Stylesheet (Mapping Specification ID):	`QdocToCoBo4e.xsl`
DTD File:	`002_confirm_bod_004.dtd`
Data Mapper Procedure:	`qqxslex.p`
Mapping Parameter 1:	`rawform`
Parameter Value 1:	`xml`

Use the values in Table 11.3 when setting up records in the Interface Control Parameters and Data Mapping Parameters frames of Import Specification Maintenance (36.8.1.3) to map Q/LinQ CONFIRM_BOD messages to the XML QDoc format.

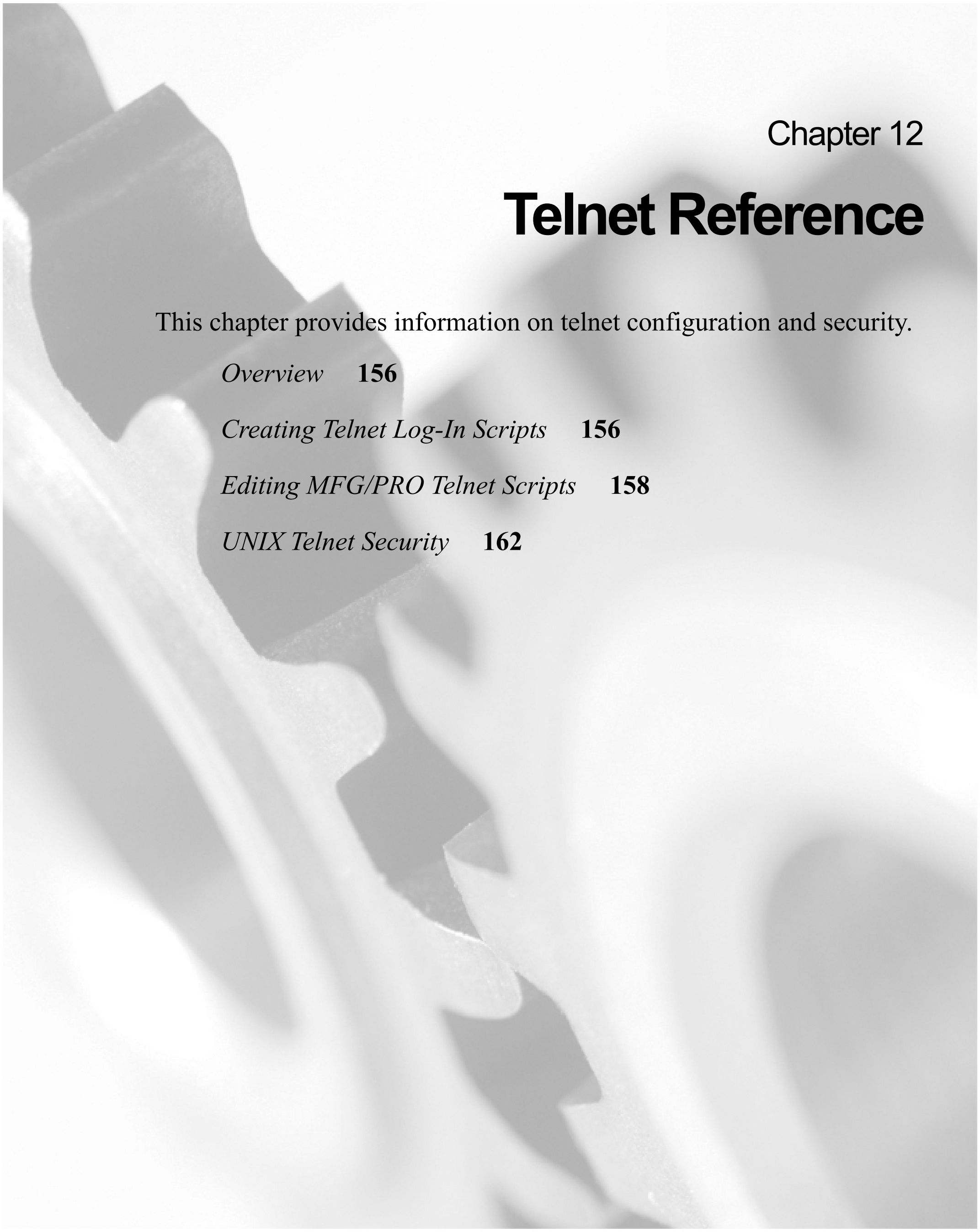
Table 11.3 Mapping Values for Importing Acknowledgements

BOD:	`CONFIRM_BOD`
XSLT Stylesheet (Mapping Specification ID):	`CoBo4iToQdoc.xsl`
DTD File:	`002_confirm_bod_004.dtd`
Data Mapper Procedure:	`qqxslin.p`
MFG/PRO Destination:	`qqprccnf.p`
Process Through User Interface:	`No`
Mapping Parameter 1:	`rawform`
Parameter Value 1:	`xml`

Section 4

Reference

This section provides reference information.



Chapter 12

Telnet Reference

This chapter provides information on telnet configuration and security.

Overview

QXtend Inbound connects to the MFG/PRO receivers using telnet. This connection is managed in the Connection Pool Manager. If QXtend Inbound and the MFG/PRO receivers are installed on the same machine, the Connection Pool Manager still uses telnet to connect the QXtend instance to the MFG/PRO instance.

During normal operation, the sequence of events is as follows:

1 Start the database servers for the MFG/PRO instances.

2 Start QXtend Inbound.

3 QXtend Inbound Connection Pool Manager creates connections to MFG/PRO. This connection is broken down into:

   a Log in to the database server.

   b Move to the MFG/PRO launch directory.

   c Launch MFG/PRO using special run scripts.

This chapter provides the necessary details on entering the correct telnet log-in script into the Connection Pool Manager, and how to create the special MFG/PRO run scripts.

Additional information is available at the end of the chapter on telnet security.

Creating Telnet Log-In Scripts

See “Add a Connection Pool” on page 97.

The value entered in the Server Startup Script field in Configuration Update Settings in the Connection Pool Manager must be the exact log-in sequence for the telnet session including both system prompts and responses.

The objective of the script is to log in to a telnet session on the database server machine and to connect to a specific running MFG/PRO database instance. Use the following sample script and descriptions to create the correct script for your environment.

Prompts and responses are separated using the pipe symbol (|). The prompt values are case sensitive. The values you enter must be identical to prompts the telnet server displays when the Connection Pool Manager attempts to log in. Verify each step manually before attempting to start Connection Pool Manager.

```
login:|QXtend|Password:|$PASSWD|$|exec ./qma.QXprod
```

`login:` is the first prompt displayed by the system.

`QXtend` is the response to this prompt, the user ID used to log in to the system.

`Password`: is the second prompt displayed.

`$PASSWD`: is the alias for the server startup password. This is used for added security measures. If you do not want the password to be displayed in clear text, then enter the password into the Server Startup Password field in the Connection Pool configuration parameters screen and this alias will be replaced by your password once it is decrypted. Hard coding the actual password in place of this alias will also function; however the password will be clearly visible to all.

For password entry instructions see “Adding a UI API Pool” on page 98.

`$` is the prompt displayed after successful log-in, indicating the system is ready.

`exec ./qma.QXprod` is the command to execute the connection script on a UNIX system. The `exec` is required to ensure that the script does not open in a new window.

`qma.QXprod` script is a copy of the standard `production` script, with the `mfgwrapper` and `apimode` parameter added to the appropriate command line.

See the next section for details.

On Windows systems, an equivalent log-in script would look like:

```
login:|QXtend|password:|$PASSWD|c:\mfgsvr\telnet>|
startmfg.bat
```

Editing MFG/PRO Telnet Scripts

See the "Generating Scripts" heading in the Character Client Install chapter of your MFG/PRO installation guide.

The telnet log-in script entered in the Connection Pool Manager calls the telnet connection script for a specific instance of MFG/PRO. The telnet connection scripts are first generated by MFG/UTIL during the installation of MFG/PRO as `qma.`*`DBSetName`* scripts, where *`DBSetName`* is the name of the database to which the script connects. For example, the telnet connection script for the Production database set is `qma.Production`.

In order for the QXtend Connection Pool Manager to connect to an MFG/PRO instance using the `qma` script for a database, modifications to the script are required.

The telnet connection scripts are located in your MFG/PRO database server administration directory.

In general, the connection script performs the following functions:

- Changes to the user's home directory
- Sets environment requirements and variables
- Provides connection parameters for the databases in the set

Sample Connection Manager Telnet Script

The following is a UNIX `qma` script. The bold sections are discussed in greater detail below.

```
#!/bin/sh
# Script to start multi-user session of MFG/PRO
# tokens:
# &DLC = Progress Directory
# &CLIENT-DB-CONNECT = command line to connect to each db in dbset
stty intr '^c'
DLC=${DLC:-/apps/progress/91d};export DLC
PATH=$PATH:$DLC;export PATH
PROMSGS=$DLC/promsgs;export PROMSGS
PROTERMCAP=$DLC/protermcap;export PROTERMCAP
PS1='$$ ';export PS1
PROPATH=${PROPATH:-.,/mfgsvr/qxtendserver,/mfgsvr,/mfgsvr/bbi}
    ;export PROPATH
# Set terminal type.
if [ ${TERM:-NULL} = NULL ]
then
        echo
        echo "Please enter your terminal type: c"
        read TERM
        export TERM
```

```
fi
#
# Start MFG/PRO.
#
cd # change to home directory
# exec $DLC/bin/_progres &DB etc
exec $DLC/bin/_progres
    /mfgsvr/db/mfgprod -ld qaddb -znotrim -trig triggers
      -db /mfgsvr/db/hlpprod -ld qadhelp
      -db /mfgsvr/db/admprod -ld qadadm
      -d mdy -yy 1920 -Bt 350 -c 30 -D 100 -mmax 3000 -nb 200
      -s 63 -noshvarfix
      -p mfwb01aa.p
      -param mfgwrapper=true,apimode=true
```

PROMSGS

Regardless of the language you are using with your MFG/PRO installation, the telnet script should reference the US English PROMSGS file. These messages are not normally displayed to the user, but QXI needs to be able to process internal messages correctly.

If you have installed more than one language, ensure the PROMSGS variable points to the US English version.

PROPATH

Add the QXtend Inbound installation directory in front of the standard PROPATH. In the example below, `/mfgsvr/qxtendserver` is the QXtend Inbound installation directory.

```
PROPATH=${PROPATH:-.,/mfgsvr/qxtendserver,/mfgsvr,/mfgsvr/bbi}
    ;export PROPATH#
```

Database Connection Parameters

The sample connection script on page 158 is a local host connection—the QXtend Inbound and the MFG/PRO receiver are on the same machine. You can also connect to MFG/PRO on a separate database server using a client/server connection.

Local Host Connection Parameters

If QXtend Inbound is on the same server as the MFG/PRO databases, you can use local host or shared memory connections to access the databases. This can improve QXtend Inbound performance by eliminating network overhead between the processes and the databases.

To enable local host connections, the QXtend Inbound parameter files must contain the connection parameters described in Table 12.1.

Table 12.1 Local Host Connection Parameters

Parameter	Description
`-db`	The physical database name parameter. Follow this parameter with the path and physical name of an MFG/PRO database.
`-trig`	The triggers parameter. Follow this parameter with the name of the directory containing the database triggers for the main MFG/PRO database. For MFG/PRO, this is the `triggers` subdirectory. The parameter value is: `-trig triggers`. This parameter is used only with main databases.
`-ld`	The logical database name parameter. Follow this parameter with the logical name of an MFG/PRO database. This parameter is used only with support databases.

Client/Server Connection Parameters

If QXtend Inbound is on a different machine than the MFG/PRO databases, use client/server connections to access the databases. This can improve QXtend Inbound system performance by spreading the MFG/PRO and QXtend Inbound resource requirements between two servers.

To enable the server processes to make client/server connections, the QXtend Inbound parameter files must contain the connection parameters described in Table 12.2.

Table 12.2 Client/Server Connection Parameters

Parameter	Description
`-db`	The physical database parameter name. Follow this parameter with the physical name of the MFG/PRO database. You do not need to include the path to the database file.
`-trig`	The triggers parameter. Follow this parameter with the name of the directory containing the database triggers for the main MFG/PRO database. For MFG/PRO, this is the `triggers` subdirectory. The parameter value is: `-trig triggers`. This parameter is used only with main databases.
`-ld`	The logical database name parameter. Follow this parameter with the logical name of an MFG/PRO database. This parameter is used only with support databases.
`-H`	The host name parameter. Follow this parameter with the machine name or IP address of the MFG/PRO database server. This guide uses *`DBServer`* as an example in place of an actual machine name or IP address.
`-S`	The database service name parameter. Follow this parameter with an MFG/PRO database service name. You can use the Database Set Maintenance utility in MFG/UTIL to find the service name for a database.
`-N`	The network parameter. Follow this parameter with the network protocol used to connect to the MFG/PRO databases. For MFG/PRO, this is TCP/IP. The parameter value is: `-N tcp`.

A Progress on UNIX example of a client/server connection would look like:

```
-db mfgprod -ld qaddb -H svr01 -S prod-srv -N tcp -trig triggers
-db mfghelp -ld qadhelp -H svr01 -S help-srv -N tcp
-db mfgadmin -ld qadadm -H svr01 -S admin-srv -N tcp
```

Additional Parameters

Two additional changes are required for QXtend Inbound implementation: changing the Progress calling program and adding the Connection Pool Manager parameters.

Progress Calling Program

The Progress program called to connect to MFG/PRO for QXtend is `mfwb01aa.p`. Change the `-p` value to reflect this.

```
-p mfwb01aa.p
```

Connection Pool Manager Parameters

The Connection Pool Manager for QXtend Inbound requires two additional parameters at the end of the connection script:

```
-param mfgwrapper=true,apimode=true
```

UNIX Telnet Security

Because QXtend Inbound communicates over telnet via HTTP, the account log-in ID and password are sent using unencrypted text. Since this may compromise system security, you should configure the telnet environment with server-side security measures in mind.

A range of security options exists to solve the unencrypted log-in and password problem. This section outlines two sample security setups: one providing a maximum level of security and one providing less security but more flexibility for Progress client session and home directory access. In both setups, it is recommended that you use a restricted shell (rsh).

Restricted Shells

Restricted shells are restricted versions of the common UNIX Bourne shell or Korn shell. In the Bourne shell, the restricted shell is run as `rsh` (`/usr/lib/rsh`), while in the Korn Shell it is run as `rksh` (`/usr/bin/rksh`).

The restricted versions of these shells allow users to log in with restricted access. They cannot:

- Use the `cd` command to change directories.
- Specify a path or command using `/`.
- Use redirection (>, >>).
- Set the value of `PATH`.

Note A user's path should not include `/usr/bin`. This lets the user run another shell, thereby inheriting access to any commands that the child shell allows. The default shell for a user is located in the `/etc/passwd` file.

Examples of Security Measures

Case 1: Maximum Security

One UNIX account with the following characteristics is used for all QXtend Inbound telnet sessions:

- No write permissions to home directory. Temporary files are written elsewhere.
- PATH, DLC, and PROPATH environment variables are set in `.profile` and inaccessible to the user.
- Startup command and/or scripts run from `.profile`.
- Telnet disconnects immediately after the user exits the MFG/PRO session.

Use the following instructions to set up Case 1:

1. Create the unique QXtend Inbound account for log in to UNIX through telnet.
2. Make the default shell for this account the restricted shell.
3. Remove all write permissions for this user in their home directory. Use the `-T` option in the remote script to specify an alternate temporary directory.
4. Set up the `.profile` to set minimal environment variables.
5. Set up the `.profile` to run the script automatically.

Example `.profile` for Case 1:

```
/*Sample .profile for QXtend session, single QXtend login*/
#set default for error (STOP) condition handling
stty intr ^C
#set environment variables
PATH=/dlc91:/dlc91/bin
DLC=${DLC - /dlc91}
PROEXE=${PROEXE - $DLC/bin/_progres}
export PATH DLC PROEXE
#Autorun remote script for QXtend access and automatically exit
exec remote.script
exit
```

Case 2: Less Security

Users have their own unique log in and password, but run the restricted shell by default:

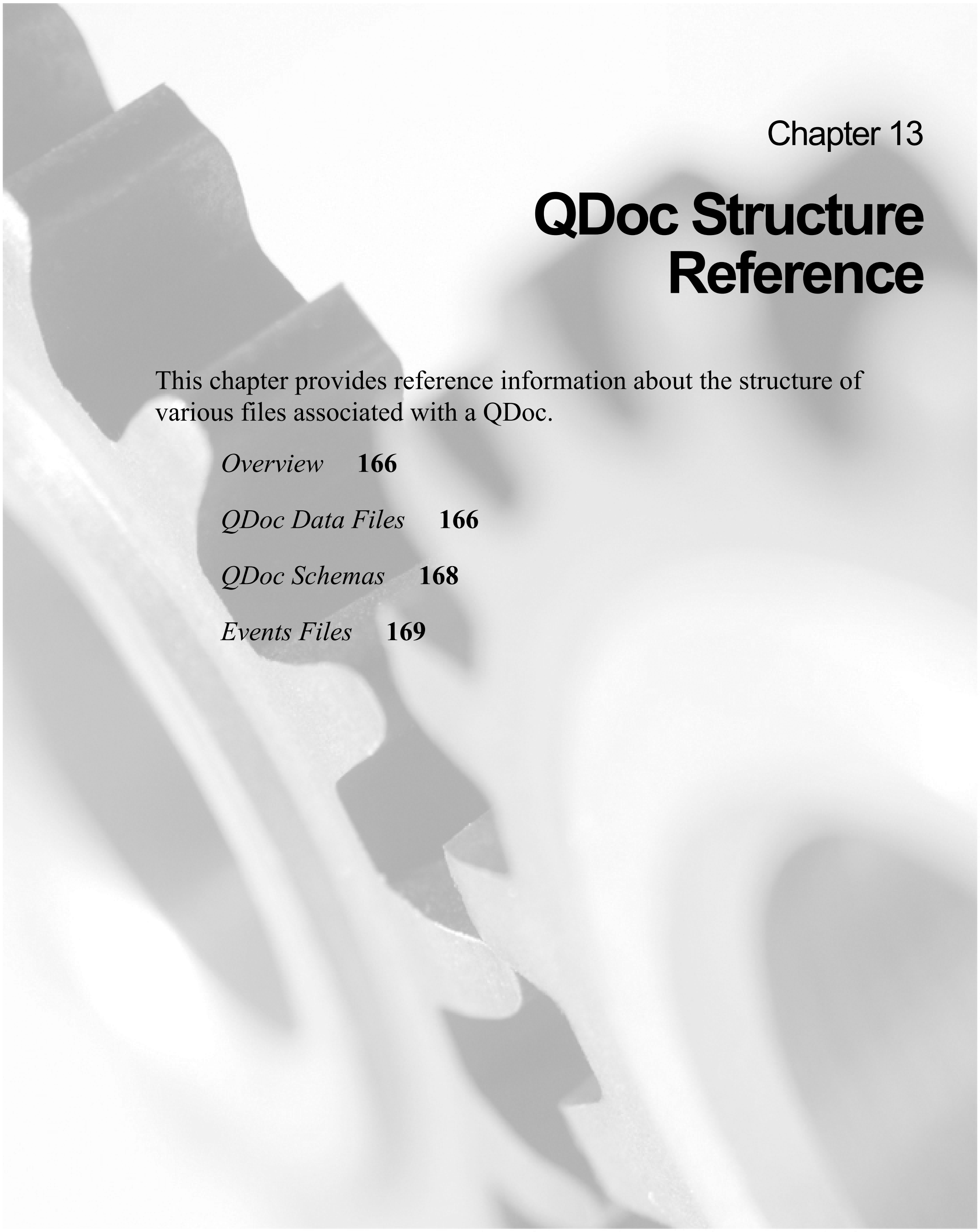
- Write permission to directory is possible, but not necessary.
- PATH, DLC, and PROPATH environment variables are set in `.profile` and inaccessible to the user.
- Users run a subset of UNIX commands, which you add to `/usr/rbin`.
- Users can run MFG/PRO manually from command line or script.

Use the following instructions to set up Case 2:

1. Create or modify accounts for users of MFG/PRO maintenance programs by changing their default shell in the `/etc/passwd` file to the restricted shell.
2. Create the directory `/usr/rbin` and copy the UNIX commands necessary for these users. Make the `/usr/bin` directory read-only so users cannot change path variables.
3. Set up a special `.profile` for the maintenance program users.
4. Set the minimal environment variables, remembering to include `/usr/rbin`.
5. Copy the QXtend Inbound telnet connection script to each user's home directory with read-only access.
6. Put any other necessary read-only script files in the home directory.

Example `.profile` for Case 2:

```
#/* Sample .profile for QXtend session, for individual logins
#remote.script should be in home dir; executable by QXtend*/
#set default for error (STOP) condition handling
stty intr ^C
#set environment variables
PATH=/dlc91:/dlc91/bin:/usr/rbin#don't forget /rbin directory
DLC=${DLC - /dlc91}
PROEXE=${PROEXE - $DLC/bin/_progres}
export PATH DLC PROEXE
```



Chapter 13

QDoc Structure Reference

This chapter provides reference information about the structure of various files associated with a QDoc.

Overview

QDocs are inbound, XML, SOAP-compliant, data documents for target MFG/PRO instances. A QDoc is not accepted by QXtend Inbound, however, unless it validates against two schema documents, and it is not entered into MFG/PRO unless an events file exists for the target program.

In addition, QDoc schemas and events files are generated in the QGen utility from a QGen-built data file.

The following sections are intended to provide a sample of each of the various QDoc components—data file, QDoc schema, and target program events file.

QDoc Data Files

See Chapter 8, "Using QGen," on page 111 for details.

The QDoc data file is generated in QGen. The following is an excerpt of the `maintainSalesOrder.dat` file. The bold numbers are QGen sequence numbers and are the beginnings of lines. Lines where three periods appear (...) show where lines have been removed for simplicity. A description of relevant document contents follows the file excerpt.

```
1000,"salesOrder",1,"Start/End Iteration","f4","a","so_nbr",
"soNbr#0:a021","soNbr","Order",yes,no,0,"x(8)","CHARACTER",
"so_mstr","RETURN",yes,no,no,no,"","","",yes,yes,
"iedPortTransh#0:a051"
2000,"",1,"None","","a","so_cust","soCust#0:a021","soCust",
"Sold-To",no,no,0,"x(8)","CHARACTER","so_mstr","RETURN",yes,no,
no,no,"","","",yes,yes,""
3000,"",1,"None","","a","so_bill","soBill#0:a021","soBill",
"Bill To",no,no,0,"x(8)","CHARACTER","so_mstr","RETURN",yes,no,
no,no,"","","",yes,yes,""
4000,"",1,"None","","a","so_ship","soShip#0:a021","soShip",
"Ship-To",no,no,0,"x(8)","CHARACTER","so_mstr","RETURN",yes,no,
no,no,"","","",yes,yes,""
4100,"",1,"None","",?,"yn","yn#0:1","yn",?,no,no,0,"yes/no",
"LOGICAL",?,"RETURN",yes,no,no,no,"","","",yes,yes,""
4200,"",1,"None","","new_ship","ad_name",
"adName#0:new_ship041","adName","Name",no,no,0,"x(28)",
"CHARACTER","ad_mstr","RETURN",yes,no,no,no,"","","",yes,yes,""
4300,"",1,"None","","new_ship","ad_line1",
"adLine1#0:new_ship041","adLine1","Address",no,no,0,"x(28)",
"CHARACTER","ad_mstr","RETURN",yes,no,no,no,"","","",yes,yes,""
...

58000,"transComment",2,"Start/End Iteration","f4","cmmt01",
"cd_ref", "cdRef#0:cmmt01021","cdRef","Master Reference",no,no,
0,"x(40)","CHARACTER","cd_det","return",yes,no,no,no,"","","",
yes,yes,"cdRef#0:cmmt01021"
```

```
59000,"salesOrderDetail",2,"Start/End Iteration","f4:f4","c",
"line","line#0:c051","line"," Ln",yes,no,0,">>9","INTEGER",?,
"return",yes,no,yes,no,"f4","S","return",yes,yes,
"cdType#0:cmmt01021"
59100,"allocationDetail",3,"Start/End Iteration","f4","alloc",
"lad_loc","ladLoc#0:alloc1091","ladLoc","Location",yes,no,0,
"x(8)","CHARACTER","lad_det","RETURN",yes,no,no,no,"","","",
yes,yes,"ladQtyAll#0:alloc1091"
59200,"",3,"None","","alloc","lad_lot","ladLot#0:alloc1091",
"ladLot","Lot/Serial",no,no,0,"x(18)","CHARACTER","lad_det",
"RETURN",yes,no,no,no,"","","",yes,yes,""
...
68400,"features",3,"Start/End Iteration","f4","w",
"work2_feature","work2Feature#0:w491","work2Feature","Feature",
yes,no,0,"->>>,>>9.9<<<<<<","CHARACTER","work2_list","RETURN"
,yes,no,no,yes,"","","",yes,yes,"work2NetPrice#0:w091"
68500,"configuredItemDetail",4,"Start/End Iteration","f4",
"w","work2_comp","work2Comp#0:w091","work2Comp","Item Number"
,yes,no,0,"->>>,>>9.9<<<<<<","CHARACTER","work2_list","RETURN"
,yes,no,no,yes,"","","",yes,yes,"work2NetPrice#0:w091"
...
145000,"",2,"None","","a","ied_region","iedRegion#0:a051",
"iedRegion","Region of Origin/Dest",no,no,0,"x(3)","CHARACTER"
,"ied_det","RETURN",yes,no,no,no,"","","",yes,yes,""
146000,"",2,"None","","a","ied_port_transh","iedPortTransh#0:a0
51","iedPortTransh","Port of Transshipment",no,no,0,"x(8)",
"CHARACTER","ied_det","RETURN",yes,no,no,no,"","","",yes,yes,""
```

Each sequence number represents a field. The sequence numbers reflect the program hierarchy. The numbers ascend in increments of 1000 if the fields are in their own frames or require the Go key to proceed. At 4000, where a separate frame is encountered, the fields within the frame ascend in increments of 100. A frame encountered inside this frame would increment in 10's, and another inside that would increment in 1's.

The first entry (1000) includes a name (salesOrder), the iteration (1), and the identifier (Start/End Iteration). It also includes the key required to leave the iteration (F4).

Other entries do not include names until 58000. This is the standard Transaction Comments entry. It starts and ends an iteration, and so is iteration number 2. The next entry, 5900, named salesOrderDetail, is also iteration 2. This is because the comments iteration is already complete by the time you reach this field. The allocationDetail iteration, 59100, and all the fields associated with it, is numbered iteration 3 because it occurs inside the salesOrderDetail iteration.

The other iterations in the program—features and configuredItemDetail— are shown. Then there is a break until the last two entries in the file.

QDoc Schemas

QDoc schemas are a product of the QGen generate process. Two schemas are created, a base schema and a type schema. For the first version of maintainSalesOrder QDoc on eB, these would be titled:

- `maintainSalesOrder-eB_1.xsd`
- `salesOrderType-eB_1.xsd`

QGen combines the name of the QDoc with the target MFG/PRO version to create the file names.

See page 193.

Chapter 15, “QDoc Specifications and Standards,” includes detailed information about how to name QDocs and the elements included in the schema files.

Base QDoc Schema

The base QDoc schema primarily identifies the name and location of the type QDoc schema. An incoming QDoc is validated against this base schema for agreement with the existing QDoc schemas. The following is the entire base QDoc schema, `maintainSalesOrder-eB_1.xsd`.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <schema xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:enc="http://www.w3.org/2002/12/soap-encoding"
    xmlns:qdoc="http://www.qad.com/qdoc/eb"
    xmlns:qcom="http://www.qad.com/qdoc/common"
    targetNamespace="http://www.qad.com/qdoc/eb"
    xml:lang="EN">
      <include schemaLocation="salesOrderType-eB_1.xsd" />
    - <element name="maintainSalesOrder">
        - <complexType>
        - <sequence>
            <element name="salesOrder" type="qdoc:SalesOrderType"
             minOccurs="1" maxOccurs="unbounded" />
            </sequence>
            <attributeGroup ref="qcom:commonAttributes" />
            </complexType>
      </element>
  </schema>
```

Type QDoc Schema

The type QDoc schema contains the program hierarchy and the field and data requirement details. Lines where three periods appear (...) show where lines have been removed for simplicity. A description of relevant document contents follows the file excerpt.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <schema targetNamespace="http://www.qad.com/qdoc/eb"
    xmlns:qcom="http://www.qad.com/qdoc/common"
```

```
  xmlns:qdoc="http://www.qad.com/qdoc/eb"
  xmlns:enc="http://www.w3.org/2002/12/soap-encoding"
  xmlns="http://www.w3.org/2001/XMLSchema" xml:lang="EN">
    <include schemaLocation="qdocCommon-eB_1.xsd" />
  - <complexType name="SalesOrderType" qdoc:primaryKeys="soNbr">
  - <complexContent>
  - <extension base="qdoc:ApiTempTableType">
      - <sequence>
        <element name="operation" type="qdoc:OperationType" minOccurs=
      "0" />
      - <element name="soNbr" type="string" minOccurs="0">
      - <annotation>
        <documentation>Order</documentation>
        </annotation>
        </element>
      - <element name="soCust" type="string" minOccurs="0">
      - <annotation>
        <documentation>Sold-To</documentation>
        </annotation>
      ...
      </element>
      - <element name="ladQtyAll" type="decimal" minOccurs="0">
      - <annotation>
        <documentation>Qty Alloc</documentation>
        </annotation>
        </element>
        </sequence>
    </extension>
    </complexContent>
    </complexType>
</schema>
```

Each element is a field in the target MFG/PRO program. No validation is required in the XML file because the validation in MFG/PRO remains active during a QDoc entry session.

Events Files

Events files contain navigation requirements for a target MFG/PRO program and by default are named the same as the MFG/PRO procedure with an `.xml` extension. The sales order entry program in MFG/PRO is `sosomt.p`; the events file for Sales Order Maintenance is `sosomt.xml`.

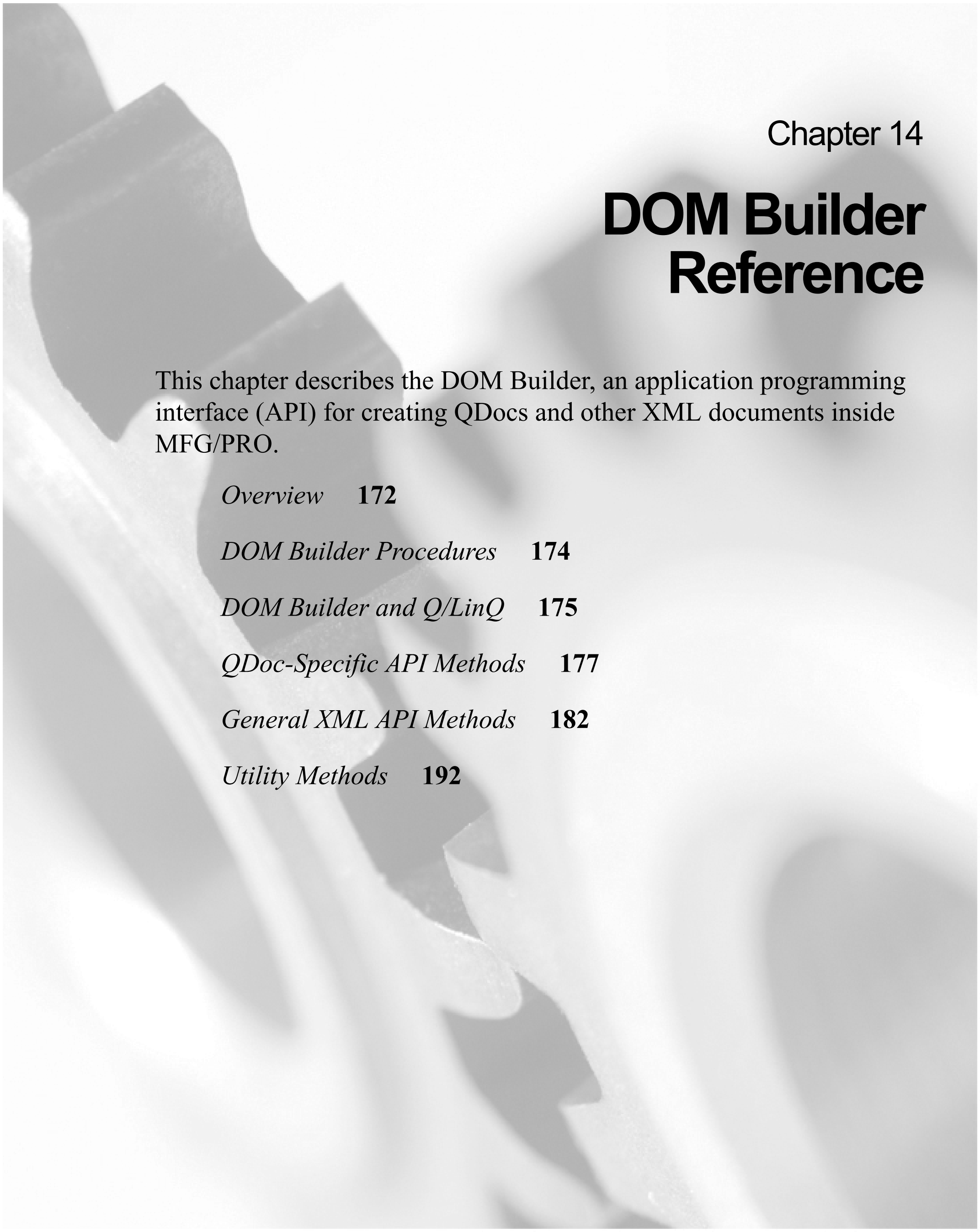
The following is the entire `sosomt.xml` file. A description of relevant document contents follows the file.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <apicontroller xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://ieli08.qad.com/apimodel
  /xmlenginexml/events/apicontroller.xsd">
    <program name="String" mfgprobase="String" delta-version="String"
    description="String" default-operation="M" />
  - <iterations>
      - <field uid="soNbr#0:a02" name="so_nbr" inheritevents="true">
      <IterationEvent iterationname="salesOrder" exititeration="f4" />
      </field>
      - <field uid="cdRef#0:cmmt0102" name="cd_ref"
         inheritevents="true">
      <IterationEvent iterationname="transComment" exititeration="f4" /
      >
      </field>
      - <field uid="line#0:c05" name="line" inheritevents="true">
      <IterationEvent iterationname="salesOrderLineDetail"
```

```
        exititeration="f4:f4" />
        </field>
        - <field uid="work2Comp#0:w09" name="work2_comp" inheritevents=
        "true">
        <IterationEvent iterationname="configurationDetail"
        exititeration="f4:f4" />
        </field>
        - <field uid="ladLoc#0:alloc109" name="lad_loc" inheritevents=
        "true">
        <IterationEvent iterationname="allocationDetail" exititeration=
        "f4" />
        </field>
        - <field uid="cmtSeq#0:cmmt0102" name="cmt_seq" inheritevents=
        "true">
        <IterationEvent iterationname="transComment" exititeration="f4" /
        >
        </field>
        - <field uid="taxLine#0:b05" name="tax_line" inheritevents=
        "true">
        <IterationEvent iterationname="taxDetail" exititeration="f4" />
        </field>
        - <field uid="iedLine#0:a05" name="ied_line" inheritevents=
        "true">
        <IterationEvent iterationname="orderLineDetail" exititeration=
        "f4" />
        </field>
        </iterations>
        - <operation action="R">
        - <events>
        - <field uid="soDueDate#0:b012" name="so_due_date" inheritevents=
        "false">
        <DeleteEvent promptfield="delYn#0" sendvalue="y" actionkey=
        "return" />
        </field>
        - <field uid="line#0:c05" name="line" inheritevents="true">
        <FirstEntryEvent prekey="f4" sendvalue="S" postkey="return" />
        </field>
        </events>
        </operation>
        - <operation action="A,M,S">
        - <events>
        - <field uid="line#0:c05" name="line" inheritevents="true">
        <FirstEntryEvent prekey="f4" sendvalue="S" postkey="return" />
        </field>
        </events>
    </operation>
</apicontroller>
```

The first set of entries in the events file are the iterations. The first iteration starts on the field so_nbr for the iteration salesOrder. The second iteration is the transaction comments iteration in the same order as in the data file. Each iteration is identified by MFG/PRO field, iteration name, and the key required to exit the iteration.

Following the iterations is the delete event and the first entry event fields. Delete fields require a field name, the value to send to confirm the deletion, and the key to complete the deletion.



Chapter 14

DOM Builder Reference

This chapter describes the DOM Builder, an application programming interface (API) for creating QDocs and other XML documents inside MFG/PRO.

Overview

Extensible Markup Language (XML) is a widely used standard grammar for exchanging business documents between applications, especially over communications networks. MFG/PRO defines QDocs—XML documents containing MFG/PRO application data that conforms to a QAD-specified syntax—for this purpose.

Many interoperability scenarios require QDocs or other XML documents to be generated within MFG/PRO and exported, perhaps as a result of some interesting business event that updates the MFG/PRO database. The purpose of the DOM Builder API is to provide MFG/PRO software developers with a high-level set of callable procedures that can be used to create such XML documents within the MFG/PRO run-time environment, hiding many of the details of QDoc and XML syntax from the programmer.

The name DOM Builder comes from the term *Document Object Model (DOM)*, the major industry-standard API for building XML documents. DOM objects are tree-representations of XML documents that contain the parent-child node relationships of XML parent elements, child elements, and attributes and values, letting the software developer navigate from parent nodes to child nodes or vice-versa using designated API methods.

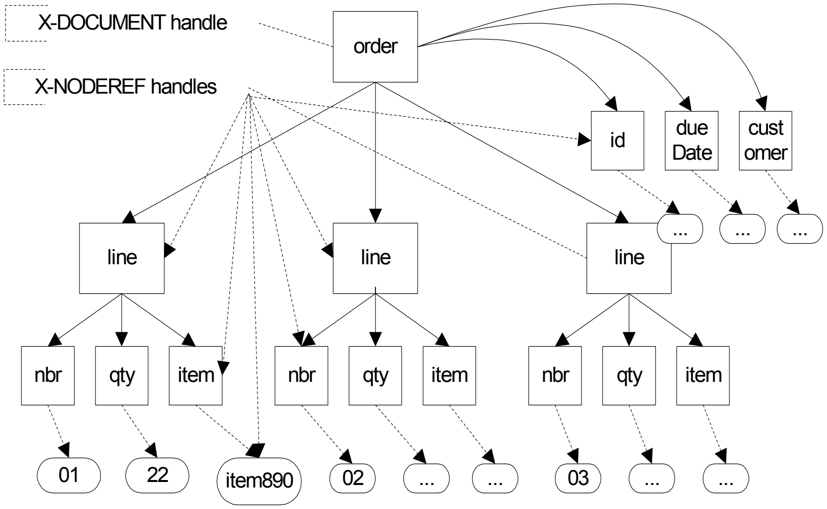
Beginning with Progress 9, the Progress 4GL contains language elements that allow XML documents to be represented and manipulated in memory as DOM objects. These XML objects may be constructed, navigated, and parsed node-by-node using particular Progress 4GL methods. The objects are referenced in Progress source code as X-DOCUMENT and X-NODEREF handles.

The following sample illustrates a simple XML document and its partial DOM representation. Figure 14.1 is a graphical representation of this code sample.

```
<order>
    <id>ord1234</id>
    <dueDate>2003-08-06</dueDate>
    <customer>cust567</customer>
    <line>
        <nbr>01</nbr>
        <qty>22</qty>
        <item>item890</item>
    </line>
    <line>
```

```
        <nbr>02</nbr>
        <qty>7</qty>
        <item>item123</item>
    </line>
    <line>
        <nbr>03</nbr>
        <qty>32</qty>
        <item>item456</item>
    </line>
</order>
```

Fig. 14.1
XML Document and DOM Representation



Using the DOM Builder, MFG/PRO programs can:

- Assemble MFG/PRO data to be placed into an XML document in either character variables or temp-table form.
- Send the information to the DOM Builder one level or one node at a time.
- Obtain the handle of the resulting XML parent node.

Once completed, the entire XML document can be retrieved as either a handle, temp-table, or character value. The MFG/PRO developer does not have to directly manipulate each XML node using the lower-level XML methods of Progress, as the DOM Builder does so automatically.

DOM Builder Procedures

The DOM Builder is a group of API methods written as internal procedures in two MFG/PRO programs that are run persistently.

- General XML DOM Builder: `gpdomcr.p`
- QDoc DOM Builder: `gpqdoccr.p`

The XML DOM Builder is a super-procedure of the QDoc DOM Builder, as the latter reimplements some of the general XML methods to automatically add attributes that appear in all valid QDocs, such as the QAD-defined XML namespaces. The QDoc DOM Builder should be started as persistent from MFG/PRO if the objective is to generate QDocs. It will automatically launch the XML DOM Builder.

See "QDoc-Specific API Methods" on page 177.

If the objective is to generate non-QDoc XML, the general XML DOM Builder should be started as persistent instead. Most of the methods are applicable to both QDocs and other XML documents, but some of the methods are QDoc-specific.

Most programs that use the DOM Builder API would structure their method calls as follows:

- Start `gpqdoccr.p` or `gpdomcr.p` running persistently, depending on whether the intent is to create QDocs or other kinds of XML documents.
- Call `createNewXMLDocument` to create the XML document.
- Call `addNode`, `addNodeGroup`, `addRecordNode`, `addAttributes`, and `createException` in combination in order to populate the XML document level-by-level, node-by-node.
- Call `getXMLAsTempTable`, `getXMLAsDOM`, `getXMLAsString`, or `getXMLAsFile` to retrieve the completed XML document objects in the form required.
- Call `deleteDocument` to delete the XML objects.

Important Only a single XML document can be under construction at one time in a single MFG/PRO session. Each time `createNewXMLDocument` is called, any XML document already in process for that session is lost and a new one started.

DOM Builder and Q/LinQ

The DOM Builder is a general-purpose API that is not part of Q/LinQ. However, it can easily be used in conjunction with the Q/LinQ publishing API to publish QDocs and other XML documents for outside applications. In Q/LinQ publishing scenarios, the DOM Builder methods would be called to create the desired XML object and serialize it into the records of a temp-table. The resulting temp-table would be passed to the qqPublishXMLDoc method of the publishing API.

See *Technical Reference: Q/LinQ* for details on Q/LinQ APIs.

The following partial code sample illustrates how the two APIs can be used together. Also, the Q/LinQ program responsible for creating QDoc confirmation documents, qqsndack.p, is a good working example.

Use of the publishing API consists of a single call to qqPublishXMLDoc, but requires that the XML document be built beforehand using the XML capabilities of the Progress 4GL. It accepts an X-DOCUMENT handle to an XML document, and serializes it into the Q/LinQ export queue.

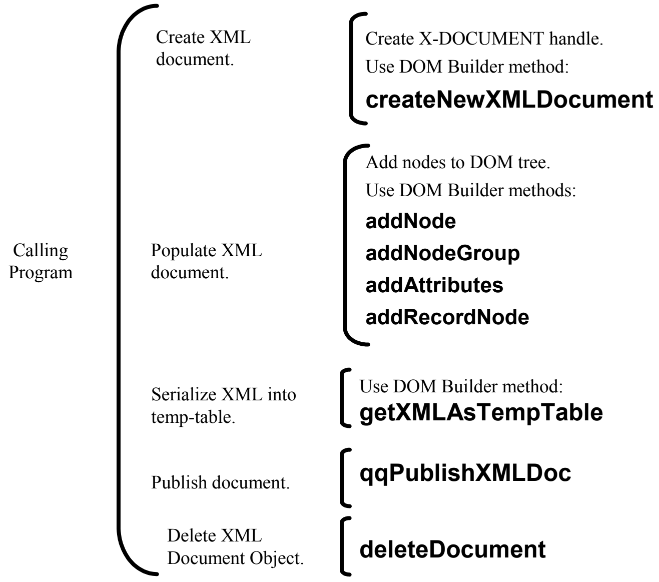
```
/*
* Use Progress XML DOM object support to build your XML document. If
* possible, call the DOM Builder methods to create the XML data and
* store it in a handle. The key methods are createNewXMLDocument,
* createNewQDoc, addNodeGroup, addRecordNode, addNode,
* addAttributes, getXMLAsTempTable.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

{pxrun.i
    &PROC = createNewXMLDocument
    &PARAM = “(
        ...)”
    &HANDLE = domBuilder
    &CATCHERROR = true
    }

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

...

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

/*Set the document type and other control information*/
```

```
/*Populate the input parameters for qqPublishXMLDoc*/
/*Call internal procedure to create document log*/

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

/*Delete the completed document to reclaim memory resources*/
{pxrun.i
    &PROC = deleteDocument
    &HANDLE = domBuilder
    &CATCHERROR = true
    }
```

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

Figure 14.2 illustrates the logic of publishing in XML format.

Fig. 14.2 Publishing API Adapter Logic for XML Documents



QDoc-Specific API Methods

This section describes the primary methods provided by the DOM Builder that are applicable only to QDocs.

createException

This method processes all exceptions present in the tmp_err_msg table, creating a QDoc exception element for each record and adding it to the in-process QDoc under the given parent element node. It is useful for building a QDoc response for an earlier QDoc request.

The tmp_err_msg temp-table is used by some MFG/PRO programs, and in particular by all MFG/PRO API procedures, to contain MFG/PRO error and warning messages generated during transaction processing. It is defined in the include file pxttmsg.i.

The following table lists parameters of createException.

Parameter Name	**Type**	**Description**	**I/O**	**Req**
errorTableBuffer	Handle	Handle to the tmp_err_msg temp-table buffer.	I	Y
parentNode	Handle	X-NODEREF object pointing to the parent node under which the exception elements will be placed as child elements.	I	Y

createNewXMLDocument

This method creates a new QDoc.

The following table lists parameters of `createNewXMLDocument`.

Parameter Name	Type	Description	I/O	Req
`qdocName`	Char	Name of the QDoc root element.	I	Y
`qdocVersion`	Char	Version attribute of the QDoc root element.	I	Y
`qdocSchemaLocation`	Char	URL of the QDoc's XML schema, for syntax validation purposes.	I	N
`sxopeTrans`	Logical	Indicates whether the entire QDoc must be processed as a single unit of work.	I	Y
`logTrans`	Logical	Indicates whether the QDoc should be logged during processing.	I	Y
`transId`	Char	User-defined identifier for the QDoc, for tracking purposes if it is logged.	I	N
`suppressResponseDetails`	Logical	Indicates whether the response to the QDoc should include detailed application data, or only the outcome of processing with any exceptions.	I	Y
`qdocRootNode`	Handle	`X-NODEREF` object pointing to the root node of the new QDoc.	O	Y

createReturnStatus

This method creates a node in the QDoc indicating the success or failure of an earlier corresponding QDoc request. It is useful for building a QDoc response for an earlier QDoc request.

The following table lists parameters of `createReturnStatus`.

Parameter Name	**Type**	**Description**	**I/O**	**Req**
`returnStatus`	Char	Value to be stored as the return status of an earlier request.	I	Y
`qdocNode`	Handle	`X-NODEREF` object pointing to the QDoc root node.	I	Y

createSOAPMessage

This method creates a SOAP 1.2 envelope with a valid QDoc SOAP header, and wraps the designated QDoc into the SOAP body. It is useful for packaging a QDoc in the form that would be required for a Web service request or response.

The following table lists parameters of `createSOAPMessage`.

Parameter Name	Type	Description	I/O	Req
`senderURL`	Char	URL that identifies the sending party or application.	I	N
`receiverURL`	Char	URL that identifies the receiving party or application.	I	N
`senderDocumentId`	Char	Locally unique identifier for the QDoc SOAP message being generated.	I	N
`documentStandard`	Char	Identifies the standard or specification to which the QDoc content conforms. Normally set to qdoc.	I	N
`documentType`	Char	Identifies the type of QDoc being created. Normally set to the QDoc root element name.	I	N
`documentVersion`	Char	Identifies the version of the QDoc being created. Normally set to the QDoc version attribute.	I	N
`confirmationLevel`	Char	Indicates whether or when a response to the created QDoc-based message is required at some later time, for confirmation purposes. Normally set to all, error, or none.	I	N
`senderDocumentRef`	Char	Reference to a business or data object in the sending application.	I	N
`receiverDocumentRef`	Char	Reference to a business or data object in the receiving application.	I	N

Parameter Name	Type	Description	I/O	Req
`timeZone`	Char	Time zone of the system creating the request, in the format of the MFG/PRO Time Zone field (tzo_mstr.tzo_label) maintained in Multiple Time Zones Maintenance (36.16.22.1) or the Time Zone field (qqc_ctrl.qqc_timezone) maintained in Q/LinQ Control (36.8.24).	I	N
`requestRootNode`	Handle	`X-NODEREF` object pointing to the root element node of the QDoc to be included in the SOAP message.	I	Y

General XML API Methods

This section describes the primary methods provided by the DOM Builder that are applicable to all XML documents, whether QDocs or not.

addAttributes

This method adds a list of attributes and the values to the specified XML element node.

Important This method is implemented as a Progress function, not an internal procedure.

Return Codes

Type	**Description**
Logical	Indicates whether the function call was successful.

The following table lists parameters of `addAttributes`.

Parameter Name	**Type**	**Description**	**I/O**	**Req**
`currentNode`	Handle	Handle of the element node to which the attributes will be added.	I	Y
`attributeNames`	Char	List of the attribute names, delimited by the ASCII 3 (Progress chr(3)) character.	I	Y
`attributeValues`	Char	List of the attribute values, delimited by the ASCII 3 (Progress chr(3)) character. The number of entries is the same as the attributeNames list, and the order of values is the same as the order of names.	I	Y

addNode

This method creates a new child element node under a given parent element node, with a specified name and value.

The following table lists parameters of `addNode`.

Parameter Name	**Type**	**Description**	**I/O**	**Req**
`nodeName`	Char	Name of the new element node.	I	Y
`nodeValue`	Char	Value of the new element node.	I	N
`parentNode`	Handle	`X-NODEREF` object pointing to the element node that will be the parent of the new element.	I	Y
`newNode`	Handle	`X-NODEREF` object pointing to the new element node.	O	Y

addNodeGroup

This method creates a group of element nodes and adds them as children under a given parent element node, each with a specified name and value.

The following table lists parameters of `addNodeGroup`.

Parameter Name	Type	Description	I/O	Req
`nodeNames`	Char	List of names of the new element nodes, delimited by the ASCII 3 (Progress chr(3)) character.	I	Y
`nodeValues`	Char	List of values of the new element nodes, delimited by the ASCII 3 (Progress chr(3)) character. The number of entries is the same as the nodeNames list, and the order of values is the same as the order of names.	I	N
`parentNode`	Handle	`X-NODEREF` object pointing to the node that will become the parent element of the new elements.	I	Y

addRecordNode

This method creates an element node with a group of child element nodes under a given parent element, where the group of created nodes is an XML representation of a buffer. The caller can specify which fields from the buffer to include in the XML or, alternatively, which fields to exclude.

The following table lists parameters of `addRecordNode`.

Parameter Name	Type	Description	I/O	Req
`nodeName`	Char	Name to give the top-level element node to be created, under which element nodes corresponding to the fields in the buffer will be added as children.	I	Y
`bufferHandle`	Handle	Handle of the buffer whose contents will be used to create child element nodes.	I	Y
`prefixToStrip`	Char	A fixed prefix of characters (for example, a database table identifier) to be stripped from the XML element names derived from the buffer field names.	I	N
`includeMode`	Logical	Indicates whether the caller-provided list of buffer fields represent data to include in or exclude from the resulting XML.	I	N
`fieldList`	Char	A list of field names from the buffer that will be included/excluded from the resulting XML, depending on the `includeMode` parameter. The list is delimited by the ASCII 3 (Progress chr(3)) character.	I	N
`parentNode`	Handle	Handle to the element node under which the generated top-level element nodes will be placed as a child, and the elements corresponding to the buffer fields will be placed as grandchildren.	I	Y
`newNode`	Handle	Handle to the top-level node created.	O	Y

createNewXMLDocument

This method creates a new XML document.

➧ See page 178.

To create a QDoc, use the QDoc-specific version of this method.

The following table lists parameters of `createNewXMLDocument`.

Parameter Name	Type	Description	I/O	Req
`initialXml`	Char	Valid XML string used as the starting point of the new XML document.	I	N
`rootNode`	Handle	`X-NODEREF` object pointing to the root node of the new XML document.	O	Y

deleteDocument

This method deletes all the XML document and node objects that have been created by the DOM Builder. It should be called by any programs using the DOM Builder after it has finished building an XML document, in order to reclaim memory and eliminate the possibility of referencing stale handles later on.

The following table lists parameters of `deleteDocument`.

Parameter Name	Type	Description	I/O	Req
None				

getXMLAsDOM

This method returns the node referencing the generated XML document.

The following table lists parameters of `getXMLAsDOM`.

Parameter Name	Type	Description	I/O	Req
`builtDocument`	Handle	`X-DOCUMENT` node pointing to the generated XML document.	O	Y

getXMLAsFile

This method serializes the content of an XML document object into ASCII characters and stores it in a designated text file.

The following table lists parameters of `getXMLAsFile`.

Parameter Name	Type	Description	I/O	Req
`xmlFile`	Char	Name of the text file in which the generated XML document will be saved.	I	Y

getXMLAsString

This method serializes the content of an XML document object into a single ASCII character string.

Important Due to Progress size limitations, the number of characters in the XML document cannot exceed 32,000. If the XML document could be larger than that, it should be retrieved using the `getXMLAsTempTable` or `getXMLAsFile` method instead.

The following table lists parameters of `getXMLAsString`.

Parameter Name	Type	Description	I/O	Req
`xmlString`	Char	Text of the generated XML document.	O	Y

getXMLAsTempTable

This method serializes the content of an XML document object into ASCII characters and stores it in the records of the `ttXMLDocument` temp-table. The document is partitioned into 15,000-character chunks, with each chunk stored in a single record of the temp-table.

Important The `ttXMLDocument` temp-table is defined by include file `gpdomcr.i`. This file must be included in any MFG/PRO program that intends to load the finished XML document into a temp-table using the `getXMLAsTempTable` method. The include file defines the temp-table used by the DOM Builder to pass the XML data back to the caller.

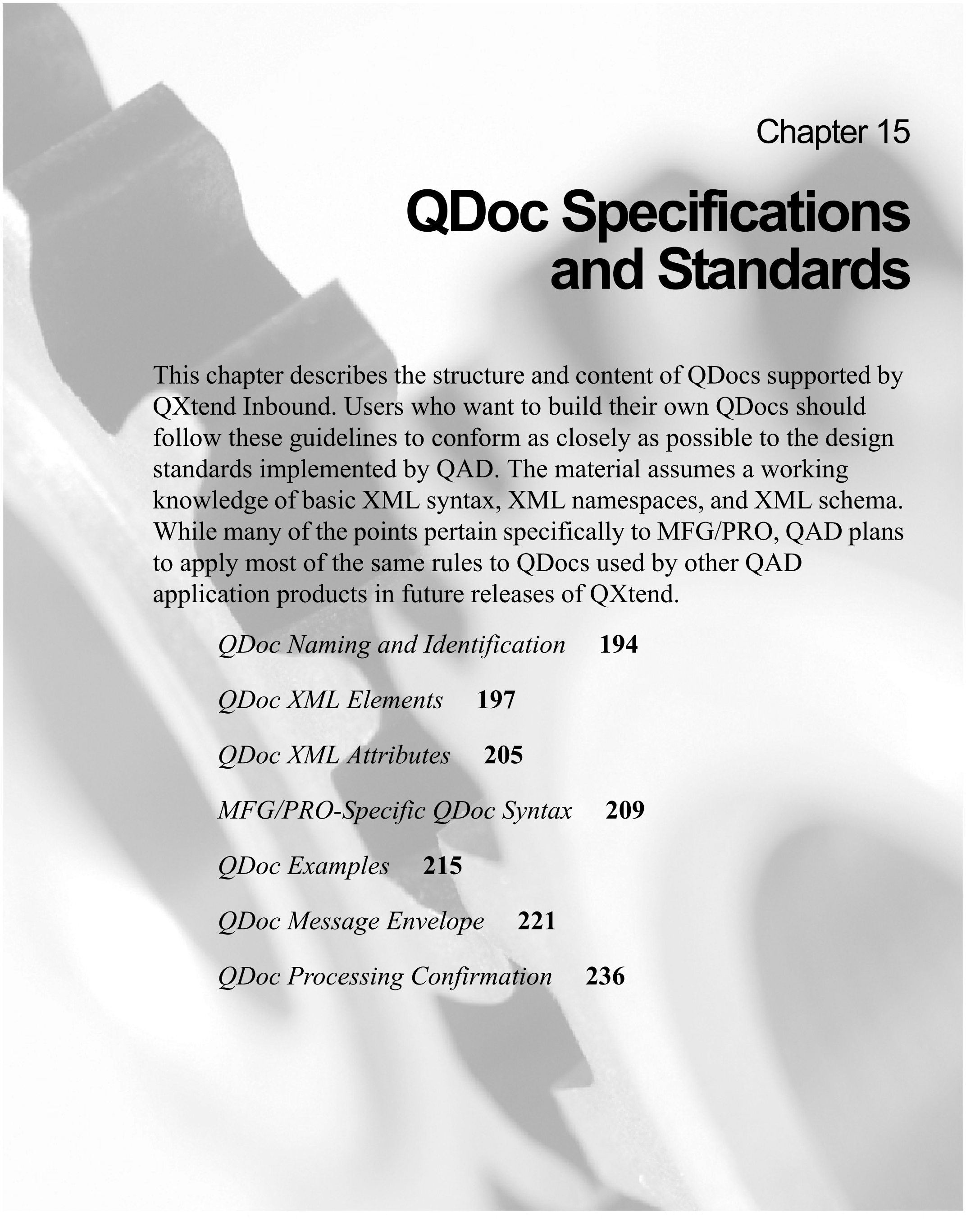
The following table lists parameters of `getXMLAsTempTable`.

Parameter Name	Type	Description	I/O	Req
`ttXMLDocument`	Temp-table	Temp-table that will contain the XML data in chunks of 15,000 characters.	I	Y

Utility Methods

In addition to the above methods, the following utility methods are also available in `gpdomcr.p.` They perform various lower-level data manipulation functions useful in the generation of XML, such as converting MFG/PRO dates and languages into the standard ISO syntax required for most XML-based standards. For detailed information, see the `gpdomcr.p` source code.

- convertDataFormat
- createNode
- getDate
- getDateTime
- getLogical
- getXMLSchemaType
- getXMLLanguage
- processArrayField
- stripPrefix



Chapter 15

QDoc Specifications and Standards

This chapter describes the structure and content of QDocs supported by QXtend Inbound. Users who want to build their own QDocs should follow these guidelines to conform as closely as possible to the design standards implemented by QAD. The material assumes a working knowledge of basic XML syntax, XML namespaces, and XML schema. While many of the points pertain specifically to MFG/PRO, QAD plans to apply most of the same rules to QDocs used by other QAD application products in future releases of QXtend.

QDoc Naming and Identification

Common QDoc Naming Convention

Each QDoc name consists of an action (verb) followed by its object (noun) that best expresses the meaning of the QDoc. The name should be as similar as possible to the corresponding API method name, if any, that implements the business logic required for the QDoc.

Table 15.1 describes QDoc naming conventions.

Table 15.1
QDoc Name Suffixes

QDoc Type	Suffix	Examples
Request In	-	maintainSalesOrder
Response Out	Response	maintainSalesOrderResponse

The suffix for the Response Out type is assigned according to present Web Services Description Language (WSDL) message-naming conventions, where requests are customarily not suffixed and responses are customarily suffixed with the literal Response.

All QDoc names are written in mixed-case or camel-case notation for Java classes, in which the first letter of each token within a name is capitalized, while all other letters are lower-case.

In the case of MFG/PRO, the `maintain` verb is used for all create-read-update-delete (CRUD) type QDocs with the detailed action (add, modify, delete, sync, get) expressed in the body of the QDoc.

Common Message Envelope

In order to route QDocs between applications, all QDocs should be wrapped inside a standard message envelope that describes the QDoc's source and destination, among other data of interest. Unlike the application-specific QDoc content with details that vary, the message envelope has a fixed, fully specified syntax that applies to all QDocs.

The envelope is designed to comply with the SOAP 1.2 standard and to translate relatively easily into emerging industry messaging standards such as Java Message Service (JMS) and Electronic Business XML (ebXML). It can be used with non-QDoc documents and must be used with QDocs.

See “QDoc Message Envelope” on page 221.

It is likely that the design of the QDoc message envelope will change in future releases, as SOAP envelope header standards are promulgated and adopted by the software industry. The QDoc envelope will evolve to become compliant with emerging Web-service standards.

Common QDoc Confirmation

A QDoc confirmation document is defined for packaging asynchronous responses to QDoc requests. The QDoc confirmation incorporates many of the elements of the QDoc message envelope in order to identify unambiguously the QDoc request that it is confirming. It is also designed to map easily into the OAGIS Confirm BOD document commonly used by ERP software providers. Like the message envelope, it has a fixed, fully specified syntax that applies to all QAD products.

See “QDoc Processing Confirmation” on page 236.

It is likely that the design of the QDoc confirmation will change in future releases, as business-to-business (B2B) data exchange standards are promulgated and adopted by the software industry. The QDoc confirmation will evolve to become compliant with emerging B2B standards.

QDoc Namespaces

All QDoc and QDoc components are defined in some XML namespace, in order to clearly identify the QAD application product with which they are associated as well as prevent QDoc naming clashes across QAD products. The namespace URIs are abstract identifiers that do not dictate the URL, directory, or path names in which the QDoc files must reside.

All QAD-standard MFG/PRO QDocs use the following namespace URI:

http://www.qad.com/qdoc/eb

QDoc components common to all QAD products use the following namespace URI. In particular, the QDoc message envelope and QDoc confirmation are defined in this namespace:

http://www.qad.com/qdoc/common

Users building custom QDocs or extending QAD-defined QDocs must define their new XML elements and attributes inside a different namespace than the QAD namespaces to allow the unambiguous mixing of QAD- and user-defined syntax within the same QDoc. Separate namespaces also allow custom QDoc syntax to be validated using separate, user-specific XML schemas.

See “version” on page 205.

Various predefined elements and attributes used across all QDoc types are defined in the namespaces and placed into a single XML schema file called `qdocCommon-<version>.xsd`. These common XML schema files do not contain application objects, but only non-application data. The appropriate version is imported into every MFG/PRO QDoc XML schema.

Schema File Names and Versioning

See “version” on page 205.

The XML schema files defining each QDoc, both requests and responses, are named `<QDoc name><Response>-<version>.xsd`, where `<version>` is the value of the QDoc version attribute. The Response component is appended on response documents only.

For all QAD products, it is expected that XML schemas will be required for each reusable data type, such as `SalesOrderType` or `PurchaseOrderLineType`, as well as for the QDocs themselves. Because the definitions of the data types will change over time, the XML schema files of the data types are versioned, with different versions of the same data type coexisting in different XML schema files. In fact, it is anticipated that most QDoc version changes will be driven by changes in the underlying data types.

See “version” on page 205.

The XML schema defining each version of a single QDoc, therefore, includes the XML schema files for the corresponding versions of its component data types. To allow the data type schemas to be referenced by version, each XML schema file defining a data type is named `<data type>-<version>.xsd`, where `<version>` is assigned in exactly the same manner as for the QDocs.

As a general rule, whenever any common data type is revised and obtains a new version, all the QDocs and/or data types containing the revised type must also obtain a new version. This practice is exactly analogous to the propagation of engineering revisions from component item numbers through parent assemblies all the way up to the end product in an engineering or manufacturing bill of material. When a component part has its version incremented, so do all its parent items.

Example A `maintainSalesOrder` version eB_1 QDoc might contain version eB_1 of the two data types `SalesOrderType` and `SalesOrderLineType`. If fields specific to eB2 are added to `SalesOrderLineType` that affect the QDoc, a version eB2_1 of `SalesOrderLineType` will be created in XML schema file `SalesOrderLineType-eB2_1.xsd`. This new file will then be included in the new QDoc XML schema file `maintainSalesOrder-eB2_1.xsd`.

Moreover, if a QDoc request schema is revised resulting in a new version, a new version of the matching QDoc response schema is also created whether or not it has actually been changed, and vice versa. Thus, any QDoc response always has the same version as its corresponding QDoc request.

QDoc XML Elements

This section describes in some detail how application data is represented in QDoc XML. The selected approach complies with the proposed SOAP 1.2 encoding standard in order to allow the QDocs to be exposed as remote procedure call (RPC) Web services in later releases.

Business Content Expressed as Elements

Following published guidelines within the XML standard and common practice, all application or business data within a QDoc is expressed as XML elements rather than attributes. Attributes are intended to represent metadata only—characteristics describing how the data in the XML document should be interpreted or processed, rather than application content. Moreover, the XML schema standard on which QAD depends for syntactic validation of QDocs provides much richer typing and extensibility features (for example, ability to inherit data type characteristics by extension or by restriction) for elements than for attributes.

Denormalized Data Associations

Many business objects require the representation of data relationships between multiple entities, not only a single entity. The most common such case is the header-detail pattern, where the lines are children with a many-to-one relationship to their parent header. This pattern is often extended to more than two levels; for example, sales order headers, lines, delivery schedules, taxes, and other special charges.

It is possible to represent such data in the form of normalized relational tables, with all associations between the entities embodied in their use of common keys; for example, an order number in the sales order line that references the associated header. It also possible to model relationships in a structured, unnormalized manner, where the child entities are explicitly contained by their respective parents with no need for common key fields.

Following common XML usage, QDocs represent parent-child relationships between entities in a structured manner, by making each instance of a child entity a component element inside a parent element. For example, a sales order is represented by a single sales order element that contains zero or more line elements, each of which in turn may contain other types of child elements.

Common Response Data

One important common QDoc requirement is the need to report the outcome of a request along with any exception conditions as part of every outbound QDoc response. Because of its inherent commonality, a single set of XML elements is specified for all QAD products.

Two required elements are included in all QDoc response documents as single-level children under the root:

- `result`, a SOAP-standard element defined in the namespace http://www.w3.org/2002/12/soap-rpc
- `returnValue`, a QAD-defined element returning the value success, warning, or error

This is pessimistic rather than optimistic usage. In the case of a multi-part QDoc (for example, a maintainSalesOrder request that includes two sales orders), the returnValue element is set to error if any of the lines in either sales order fail for any reason.

The result element always has the value returnValue, as its only purpose within the SOAP 1.2 standard is to reference the element within the SOAP response that contains the return value of the QDoc. This indirect approach permits the standard result element to be completely defined within the soap-rpc namespace, while still permitting the application to return its own strongly typed value.

Zero or more exception elements are included in all QDoc response documents as single-level children under the root. Each exception includes the following elements of the type string:

- number: identifies the type of exception (required)
- description: briefly describes the exception condition (required)
- severity: designates the severity of the exception—informational, warning, error (optional, default error)
- field: identifies the specific field in the request for which the exception was raised (optional, no default)
- context: identifies the context within the request, typically an element or row instance, in which the exception was raised (optional, no default)
- trace: a detailed trace of the method execution that raised the exception, typically a Java stack trace (optional, no default)

A common ExceptionType for the exception element is defined and stored in the QDoc namespace and common XML schema files. Its XML schema definition is shown in the following code sample.

See “QDoc Namespaces” on page 195.

It is possible for a QDoc response to contain nothing inside the QDoc root element except one or more exception elements plus the result and returnValue elements. This would be the case for errors recognized before application processing of the QDoc has begun; for example, if the QDoc request’s name-version is not supported or its XML content is not well formed.

```
<complexType name="ExceptionType">
    <annotation>
        <documentation>
        An exception message returned from a QDoc request by
        the QAD application.
        </documentation>
    </annotation>
    <sequence>
        <!-- number and description fields are required -->
        <element name="number" type="string"/>
        <element name="description" type="string"/>
        <element name="severity" minOccurs="0"
            default="error">
            <simpleType>
               <restriction base="string">
                   <enumeration value="informational"/>
                   <enumeration value="warning"/>
                   <enumeration value="error"/>
               </restriction>
            </simpleType>
        </element>
        <element name="field" type="string" minOccurs="0"/>
        <element name="context" type="string" minOccurs="0"/>
        <element name="trace" type="string" minOccurs="0"/>
    </sequence>
</complexType>
```

A common `ReturnValueType` for the `returnValue` element is also defined and stored in the QDoc namespace and common XML schema files. Its XML schema definition is shown in the following code.

```
<simpleType name="ReturnValueType">
    <annotation>
        <documentation>
        The value returned by the QAD application.
        </documentation>
    </annotation>
    <restriction base="string">
        <enumeration value="success"/>
        <enumeration value="warning"/>
        <enumeration value="error"/>
    </restriction>
</simpleType>
```

Errors Prior to Processing

Some QDoc requests cannot be processed due to syntax errors. For example, the QDoc may not be recognized because of a misspelled name; critical attributes of the QDoc required for application processing may be invalid; or the XML may be badly formed. In such cases, a normal response based on standard naming conventions is not suitable, as the identity and contents of the QDoc are being called into question.

Since QDocs are a SOAP-compliant solution, QDoc preprocessing errors are thrown back to the sender as SOAP faults.

See "SOAP Faults" on page 232.

Primitive Data Type Representations

All primitive data types used by QAD applications (character or string, integer, decimal, date, time, boolean or logical) are represented in QDocs using the data type standards specified for XML schemas. The most interesting cases in this regard are dates, times, integers, and decimals.

- Dates are represented as YYYY-MM-DD (for example, 2003-04-22).
- Times are represented as HH:MM:SS+HH:MM (for example, 23:20:06+07:00 or 23:20:06-02:00), where the expression +HH:MM is the number of hours and minutes ahead or behind Universal Time (also called Greenwich Mean Time).
- The type dateTime is an XML-schema-defined type that is a combination of the date and time types as specified by ISO 8601. Its representation is the concatenated values of date and time separated by an upper-case T, in the format YYYY-MM-DDTHH:MM:SS+HH:MM. For example, 2003-04-22T23:20:06-08:00 would denote 11:20 pm Pacific Standard Time on April 22, 2003.
- Integers are represented normally, with an optional + or – sign preceding the digits.
- Decimals are represented with the period (.) as the decimal point and optionally the comma (,) as the thousands separator.

The application is responsible for managing the display formats used in different geographies; for example:

- Reversing the usage of the period and comma for quantities
- Modifying the sequence of the day-month-year components of dates

Default, Empty, and Null Values

As a general rule, optional elements can be omitted from any QDoc if the requestor wants the target QAD application to use their default values. A particularly important special case of this rule is the modify or change operation of a CRUD request. If an element in a modify/change request is omitted, the QAD application presumes that the element's value should not be changed and retains its existing value. This approach has the major advantage of conciseness, as it permits the requestor to populate the QDoc request with only the content that is relevant and omit the rest.

However, this approach does raise the problem of ambiguity with respect to empty values. If an omitted element signifies that its default value should be used, how do you communicate a request inside a QDoc to blank out the element's value? In other words, how should an empty value be represented? The term empty instead of null is used in order to avoid confusion with the concept of a special null object/value, such as Java null or the Progress unknown value. Unlike the null value, empty values vary based on data type. For character fields empty would be a null string, for numbers the value zero, and for logicals the value false.

To distinguish between default and empty values in QDoc requests, a distinction is made in all QDocs between an omitted element and an empty-valued one. If an element is omitted, the QAD application uses its default value. If it is empty valued (that is, has the form `<element></element>` or `<element/>`), the application assumes that the element should be assigned the empty value appropriate for its type.

While XML schema allows null elements to be explicitly represented with a nil attribute, enabled by a nillable attribute in the respective elements' schema definition, use of this construct, and null values in general, is not supported by the QDoc schemas as a general rule. However, the nil and nillable attributes may be designated for use in special cases where QAD application processing requires null value input.

Array Representation

The use of arrays in API methods and interoperability documents is generally not recommended, as they can be problematic to translate into some programming languages and often reflect a poor, unnormalized data model. Nevertheless, their use is sometimes necessary to preserve compatibility with legacy application code, if for no other reason. For such cases, you can use a standard approach for representing arrays inside QDocs.

Arrays are represented according to the SOAP encoding scheme specified in SOAP 1.2 in order to better support the automatic generation of Java client stubs from the WSDL files using non-QAD tools. Various QDoc-specific limitations are imposed in order to simplify the SOAP syntax by eliminating those features not required by any QAD applications.

QDoc arrays are elements that contain multiple occurrences of only one simple element. The element for the array as a whole is named according to the same conventions as other elements in the QDoc, based on QAD application usage. The single component element is named `entry` in all QDocs regardless of usage, with `entry` defined in the QDoc common namespace so as not to conflict with the name of any QAD application array.

Every array described in a QDoc XML schema is declared as a restriction of the base type `enc:Array`. Every QDoc instance contains `itemType` and `arraySize` attributes associated with the following SOAP encoding namespace URI:

http://www.w3.org/2002/12/soap-encoding

The value of `itemType` can be any one of the following: string, boolean, decimal, int, date, time, anyType. (anyType is the theoretical supertype of all other types, somewhat like Object in Java.) The value of `arraySize` is an integer denoting the maximum size or extent of the array, or the asterisk character (*) if that value is not known. Arrays of complex elements, multi-dimensional arrays, and arrays of arrays (jagged arrays) are not allowed in QDocs.

Because SOAP 1.2 requires that arrays be included in their entirety (that is, no partial arrays), a means is needed to distinguish an array element that is being set to an empty value by a QDoc from one that is not being changed by the QDoc.

Note This is the same problem of specifying empty vs. default values except that in the case of arrays it is not possible to skip an empty element in order to designate that the default value should be used.

See “Default, Empty, and Null Values” on page 201.

In order to resolve the problem, the QDoc-defined `entry` element has one required attribute `index` and one optional attribute `skip`, also defined in the QDoc common namespace. The `index` attribute identifies each entry of the array. The skip attribute indicates whether the entry should be skipped or defaulted if set to true. If the skip attribute is set to true, the content of its containing entry element is ignored.

Syntactically, the indexed entries of the array can appear in any order. However, by convention, they should always be sequenced in ascending index order. For example, the following QDoc fragment would represent a six-member numeric array called orderQuantity containing two empty, two skipped, and two non-empty elements.

```
<qdoc:orderQuantity
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:enc="http://www.w3.org/2002/12/soap-encoding"
    xmlns:qcom="http://www.qad.com/qdoc/common"
    enc:itemType="xsi:decimal"
    enc:arraySize="6"
    >
    <!-- Entries 1, 3 are set to empty and 4, 6 are skipped -->
    <qcom:entry index="1"/>
    <qcom:entry index="2">10.583</qcom:entry>
    <qcom:entry index="3"/>
    <qcom:entry index="4" skip="true"/>
    <qcom:entry index="5">22</qcom:entry>
    <qcom:entry index="6" skip="true">ignore!</qcom:entry>
</qdoc:orderQuantity>
```

The following is the XML schema definition for this fragment, defined using SOAP conventions.

```
<schema
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:enc="http://www.w3.org/2002/12/soap-encoding"
    xmlns:qcom="http://www.qad.com/qdoc/common"
    xmlns:qdoc="http://www.qad.com/qdoc/eb">
    targetNamespace="http://www.qad.com/qdoc/eb"
    xml:lang="EN"
    >
    <import namespace="http://www.w3.org/2002/12/soap-encoding"
        schemaLocation="soap-encoding.xsd"/>
    <element name="orderQuantity">
        <complexType>
            <complexContent>
               <restriction base="enc:Array">
                   <sequence>
                       <element ref="qcom:entry"
                           minOccurs="6" maxOccurs="6"/>
                   </sequence>
                   <attributeGroup ref="enc:arrayAttributes"/>
                   <attributeGroup ref="enc:commonAttributes"/>
               </restriction>
            </complexContent>
        </complexType>
    </element>
</schema>
```

The following is the XML schema fragment defining the entry element and its index and skip attributes in the QDoc common namespace.

```
<schema
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:qcom="http://www.qad.com/qdoc/common"
    targetNamespace="http://www.qad.com/qdoc/common"
    xml:lang="EN"
    >
    <element name="entry">
        <complexType>
            <simpleContent>
               <extension base="anyType">
                   <attribute name="index" type="positiveInteger"
                       use="required"/>
                   <attribute name="skip" type="boolean"/>
               </extension>
            </simpleContent>

        </complexType>
    </element>
</schema>
```

Element Names

Except for the single root element that identifies the QDoc, a universal naming convention for QDoc elements that applies to all QAD products does not exist. Rather, product-specific conventions are used.

See "Array Representation" on page 202.

QDoc XML Attributes

All QDocs use the XML attributes described in this section.

xml:lang

This attribute is an XML standard, not QDoc-specific. It designates the national language of its associated element, using the two-letter language codes defined by the ISO 639 standard.

version

This attribute is specific to QDocs. It identifies the version of a QDoc schema to which a particular instance of that QDoc conforms, and is required at the QDoc root element level. Its format is <major product release>_<sequence>, where <major product release> is the lowest major release of the QAD application product in which the QDoc is supported (for example, eB, eB2, or eB2.1 for MFG/PRO), and `<sequence>` is an incremented sequence number within that release starting with 1. Any periods in the release name are removed.

Example The first version of an MFG/PRO QDoc that is developed to work with both the eB and eB2 releases would have the version eB_1, for both eB and eB2 users. If this QDoc were later enhanced only for releases eB3 and above, the new version would be called eB3_1, and could coexist with version eB_1 if QAD decides to support both versions concurrently. If the same enhancement were later retrofit to eB and eB2, the new version for the eB and eB2 releases would be called eB_2. Note that service packs are not treated as major releases for QDoc versioning purposes.

This versioning scheme allows multiple versions of the same QDoc to be supported concurrently for multiple product releases, while capturing in a natural way the relationships and compatibilities among the various QDoc versions and product releases.

See "QDoc Namespaces" on page 195.

For custom or user-extended QDocs, QAD suggests that custom version designators be assigned as suffixes to the QAD-supported versions on which they are based (for example, eB_2_ABC_1 for the first customization developed by ABC Company). While all customized QDocs must be defined in a namespace different from QAD's in order to avoid name-version clashes, customer-specific version identifiers help highlight their custom nature.

scopeTransaction

This attribute is an optional boolean QDoc requests at the root element level that indicates whether the processing of the QDoc should be scoped as a single transaction. In general, the units of work scoped by the application programs that process QDocs vary based on the nature of the request, the input data, and decisions made by the application designers. The `scopeTransaction` attribute allows the requestor to override normal units of work and force a single-transaction scope for the request.

In distributed environments where the requesting and receiving application are running in different application server containers or different machines, controlling the transaction scope can be inconvenient or even impossible for the requestor of a remote service. This attribute provides a simple means of controlling the transaction scope remotely.

logTransaction

This attribute is an optional boolean for QDoc requests at the root element level that indicates whether during application processing the QDoc should log the request internally for audit trail or recovery purposes. Because of the potential performance overhead associated with logging, requestors may want to avoid the cost of logging, particular in local integrations running in a stable, local network environment.

Logging is most important in distributed environments, where the QDoc application may be deployed across multiple application servers or machines using communication links that are less than 100% reliable. Even when an integration framework is receiving all QDoc requests, logging at this level may not be sufficient when the application is running as multiple components in separate environments. It may be necessary for the application to write log entries in order to protect against internal communication failures between the application component and the integration framework.

transactionID

This attribute is an optional string for QDoc requests at the root element level that designates a requestor-defined identifier. It is used only in conjunction with the `logTransaction` attribute. It is entirely requestor-defined, and does not have to be a unique key. The intent is to capture its value in any QDoc log entries, so that remote requestors can refer to an audit trail entry in case of internal communication failures.

suppressResponseDetail

This attribute is an optional string for QDoc requests at the root element level that indicates whether the business data of the QDoc response is needed, as opposed to only the exceptions raised during processing. It is provided as a performance-enhancing feature, when the requestor wants a request processed but does not need to know the results outside of the overall outcome and any exceptions encountered.

A primary example would be the processing of inbound QDocs into MFG/PRO through Q/LinQ or EDI ECommerce. As both of these products perform their transaction processing asynchronously, they need only to log success/failure and exceptions thrown, from which they may (or may not) create separate outbound QDoc confirmation documents for the ultimate requestor. Given the verbose nature of XML in general, using this attribute could measurably improve performance and/or conserve network bandwidth.

Query requests would always set this attribute to false; otherwise, the QDoc response would not include the results of the query.

Common Attribute Group

See "QDoc Namespaces" on page 195.

Several attributes—`version`, `scopeTransaction`, `logTransaction`, `transactionId`, and `suppressResponseDetail`—are used in almost all QDocs. Hence, they are defined once as members of an XML attribute group defined in common XML schema files, and referenced in all relevant QDoc definitions.

The following is the XML schema definition fragment for this group.

```
<import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="xml.xsd"/>
<attributeGroup name="qcom:commonAttributes">
    <attribute name="version" type="string"
        use="required"/>
    <attribute name="scopeTransaction" type="boolean"
        use="optional"/>
    <attribute name="logTransaction" type="boolean"
        use="optional"/>
    <attribute name="transactionId" type="string"
        use="optional"/>
    <attribute name="suppressResponseDetail" type="boolean"
        use="optional"/>
    <attribute ref="xml:lang"/>
</attributeGroup>
```

MFG/PRO-Specific QDoc Syntax

The XML-based content of all MFG/PRO QDoc requests and responses is mapped or bound at run-time to and from the native data structures of the UI, code, or JIT/S API responsible for processing. This section summarizes the fixed rules used to accomplish this binding.

If a data structure clash or mismatch is detected between the QDoc syntax and the code and JIT/S API signatures, a QDoc syntax error is thrown to the requestor and the request is not processed.

QDoc Name

The QDoc name or root element maps to the name of the API method to be run, in the case of the code and JIT/S APIs. In the case of the UI APIs, it is used to determine the MFG/PRO online program to run.

Simple Elements

Simple QDoc elements map to temp-table fields in the case of code and JIT/S APIs, or user input fields in the case of the UI APIs. Their names match the MFG/PRO names of the corresponding code and JIT/S API parameters, except that mixed-case or camel-case notation for Java classes is used in place of the underscore-delimited names common in MFG/PRO. That is, the underscore characters separating tokens within the MFG/PRO name are removed and the first letter of each token within the name is capitalized, while all other letters as well as the first letter are lower-case. For example, the MFG/PRO field pt_prod_line would be expressed as `ptProdLine` in the QDocs.

All arrays are presumed to contain simple elements. Each is mapped to a temp-table field with extent equal to the array's declared length.

Enumerated fields used across many QDocs (for example, the operation field used for CRUD processing in the code and JIT/S APIs) are defined in the common QDoc XML schema file for inclusion in all schemas.

Complex Elements

Complex QDoc elements containing other elements, representing business data entities in MFG/PRO, map to iterating frames in the case of the UI APIs and temp-tables in the case of code and JIT/S APIs. Their names are natural business terms expressed in camel-case with no database-specific notation (for example, pt_mstr for item) or special prefixes/suffixes. For elements that are associated with temp-table parameters in the code and JIT/S API methods, the QDoc element name is identical to the temp-table name but without the `tt` prefix and with the first letter changed to lower-case. For example, the temp-table `ttPurchaseOrderDet` in the `maintainPurchaseOrder` API method would be expressed as `purchaseOrderDet` in the QDocs.

Just as the code and JIT/S API temp-tables make heavy use of common field lists (for example, the include files `mfaittxt.i`, `mfctit01.i`, and so on), so do their corresponding QDoc elements. XML schema supports an inheritance mechanism whereby types can be derived by extending other types by adding fields, analogous to subclassing in object-oriented programming languages.

Using this technique, an abstract base type `ApiTempTableType` is defined that includes all the fields common to all code and JIT/S API temp-tables. Reusable derived types such as `TransCommentType` are built from `ApiTempTableType` by extension, and in turn used to build more specific types such as `PurchaseOrderCmtType`. The type hierarchy is built by inspecting the present code and JIT/S API temp-table definitions and common field lists. The collections of common data are modeled as base types in the XML schema.

See “QDoc Namespaces” on page 195.

The following XML schema definition fragment is for the `ApiTempTableType`, as well as fragments of the `TransCommentType`, `PurchaseOrderCmtType`, and `OperationType` definitions. The `ApiTempTableType` definition is stored in the XML schema file for common MFG/PRO QDoc objects, while the derived types are stored in separate schema files.

```
<complexType name="ApiTempTableType" abstract="true">
    <sequence>
        <!-- Common API temp-table fields from mfaittxt.i -->
        <element name="apiSequence" type="string"/>
        <element name="apiSuccess" type="boolean"/>
        <element name="apiExternalKey" type="string"/>
    </sequence>
```

```
</complexType>

<simpleType name="OperationType">
    <restriction base="string">
        <!-- Common Operation values from mfaiocon.i -->
        <enumeration value="A"/> <!-- Add -->
        <enumeration value="M"/> <!-- Modify -->
        <enumeration value="R"/> <!-- Remove -->
        <enumeration value="S"/> <!-- Sync -->
        <enumeration value="G"/> <!-- Get -->
        <enumeration value="N"/> <!-- No-op -->
    </restriction>
</simpleType>

<complexType name="TransCommentType">
    <complexContent>
        <extension base="qdoc:ApiTempTableType">
            <sequence>
               <!-- Common Transaction Comment temp-table fields
                   from mfctit01.i, mfctid01.i -->
               <element name="operation" type="qdoc:OperationType"
                   minOccurs="0"/>
               <element name="seq" type="int" minOccurs="0"/>
               <element name="ref" type="string" minOccurs="0"/>
               <element name="type" type="string" minOccurs="0"/>
               <element name="lang" type="string" minOccurs="0"/>
               ...
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="PurchaseOrderCmtType">
    <complexContent>
        <!-- No P.O.-specific fields to add to base Transaction
            Comments, but the type is still required in order
            to map easily to the corresponding API temp-table. -->
        <extension base="qdoc:TransCommentType"/>
    </complexContent>
</complexType>
```

Arrays

Arrays are mapped to MFG/PRO fields with extents greater than 0. As discussed earlier, only one-dimensional arrays that contain a single simple element type are allowed in QDocs. The name of the array maps to the MFG/PRO field name, subject to the formatting conventions for element names described previously. The name of all array elements is `entry` for all QDocs, defined in the QDoc common namespace.

Element Order

In general, XML schema allows the child elements under a given parent to be defined such that the element order is variable. Use of this feature would support QDoc syntax flexibility. However, it is not used for MFG/PRO QDocs because of the following constraints:

See "Complex Elements" on page 210.

- All the complex elements corresponding to code and JIT/S API temp-tables are derived by extending the abstract base type `ApiTempTable`, in order to define all the standard boilerplate temp-table fields in one place. However, the current release of XML schema forces all fields defined in the base type to precede the fields defined in the derived type, thus constraining the element order.
- The QDocs intended for code and JIT/S API processing contain primarily multi-occurrence, complex child elements corresponding to the code and JIT/S API temp-tables. However, use of variable element order underneath a given parent is allowed only if all the children are single-occurrence elements.

Required vs. Optional Elements

In general, XML schema allows the child elements under a given parent to be defined as required or optional through the `minOccurs` and `maxOccurs` attributes. The default is `minOccurs=1, maxOccurs=1`.

However, to make maintenance of the XML schema easier over time, virtually all QDoc elements are defined as optional (`minOccurs=0`). This approach also supports many common API scenarios in which a common data object is used in multiple API methods, but with context-specific validation that determines whether the elements are required or optional. It is the responsibility of the MFG/PRO application to throw exceptions if the requestor does not provide a required field using present MFG/PRO validation logic, rather than relying on XML schema syntax checking.

Normalized vs. Denormalized Representation

In the case of code and JIT/S APIs, all temp-table parameters are expressed in normalized form, in which the relationship between parent and child temp-tables is represented only as a set of common key fields (foreign keys) that occur in both tables. That is, one temp-table is not contained in or structurally subordinate to the other. Rather, all appear as siblings in the API signature.

A complex QDoc element that contains another complex QDoc element (in denormalized fashion) is mapped to two separate temp-tables in the code and JIT/S APIs. In addition, all designated key fields appearing in the parent element are duplicated inside the child temp-table records with the same values as in their respective parent temp-table records. In other words, the normalized temp-table records contain foreign key values referencing their respective temp-table parent records, as with any normalized relational data structure.

In order to designate those simple QDoc elements (fields) inside a particular complex QDoc element (temp-tables) that constitutes the primary key, a special `primaryKeys` attribute is defined in the QDoc namespace for inclusion in all QDoc XML schema files. Its definition is stored in common XML schema files.

This attribute is used with all XML schema element definitions that are bound to code and JIT/S API temp-tables, so that the QDoc Request Handler can identify the parent-child and key relationships from the XML schema and properly normalize the resulting temp-tables. The following QDoc examples illustrate use of this attribute within QDocs.

The XML schema definition for the `primaryKeys` attribute is as follows.

```
<attribute name="primaryKeys" type="token">
    <annotation>
        <documentation>
        Lists the primary keys of the QDoc element for the
        purpose of normalizing its child elements into sibling
        data structures with fully unique keys.
        Used to map the unnormalized QDoc XML into a temp-table
        representation for the code APIs.
        </documentation>
    </annotation>
</attribute>
```

The UI APIs work natively with denormalized data structures, and do not require such transformation to normalize the QDoc elements.

CRUD QDocs

QDocs that request basic create-read-update-delete (CRUD) maintenance against some MFG/PRO business object are very common. Because of their ubiquity, a single format is used for all of them along the lines of the signature standards defined for the code and JIT/S APIs.

- All QDocs requesting CRUD activity on an MFG/PRO entity are named using the verb maintain (for example, `maintainSalesOrder`, `maintainSupplier`). This terminology keeps the QDoc names relatively close to the MFG/PRO menu procedures whose functionality they incorporate.
- At the beginning of every complex XML element that denotes a business object to be maintained (for example item, `salesOrder`, `supplier`), an operation element describes the type of maintenance requested for the object. Allowed values are A (add), M (modify), D (delete), G (get), S (sync), N (no-op).
  - Sync indicates that the object should be added if not present in the database and modified if present, essentially the same way that MFG/PRO maintenance procedures currently work.
  - Add raises an error if the object already exists.
  - Modify raises an error if the object does not exist.
  - No-op is a special value signifying that the element should be skipped; it may be useful for cases in which the value of some QDoc element helps navigate to or access subsequent elements, but for which no processing is required.

Operation elements can exist at multiple levels within a QDoc, such as the order header, line, and transaction comment levels.

Attributes

For QDocs implemented by code and JIT/S APIs, the `scopeTransaction`, `logTransaction`, and `transactionId` attributes are mapped to the standard `scopeTrans`, `logTrans`, and `transId` parameters respectively. The `suppressResponseDetail` attribute has no counterpart in the code and JIT/S API call signatures.

QDoc Extensions and Customizations

Because bindings between QDocs and MFG/PRO are based on a fixed set of rules, the only way to extend or modify a QDoc is to extend or modify:

- The MFG/PRO code that implements the QDoc, and
- The XML schema for the QDoc

The new/modified QDoc elements must follow the naming and format conventions described for standard QDoc content, but must be defined within a non-QAD namespace to distinguish them from QAD-standard syntax.

If the custom QDoc is to be processed using the MFG/PRO menu procedures (the UI APIs), an XML schema for the QDoc should be generated using the QGen tool.

See Chapter 8, "Using QGen," on page 111.

QDoc Examples

MFG/PRO Inbound QDoc Request

This is a sample partial QDoc request to maintain a purchase order in MFG/PRO, based on the API method `maintainPurchaseOrder`. The QDoc message envelope is omitted, as it is described elsewhere.

See "QDoc Message Envelope" on page 221.

The examples assume that the earliest supported MFG/PRO release is eB.

```
<?xml version="1.0" encoding="UTF-8"?>
<maintainPurchaseOrder version="eB_1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:enc="http://www.w3.org/2002/12/soap-encoding"
    xmlns="http://www.qad.com/qdoc/eb"
    xmlns:qcom="http://www.qad.com/qdoc/common"
    schemaLocation="http://www.qad.com/qdoc/eb/schemas"
    scopeTransaction="true"
    logTransaction="true"
    transactionId="abc123"
    suppressResponseDetail="false"
    xml:lang="EN"
    >
    <purchaseOrder>
        <operation>A</operation>
        <nbr>PO1234</nbr>
        <vend>00000001</vend>
        <ordDate>2002-04-01</ordDate>

        <rmks>This is a test PO</rmks>
        <purchaseOrderCmt>
```

```
            <operation>A</operation>
            <seq>01</seq>
            <cmmt>Line 1 of comment!</cmmt>
        </purchaseOrderCmt>
        <purchaseOrderDet>
            <operation>A</operation>
            <line>01</line>
            <dueDate>2002-05-01</dueDate>
            <part>10-10000</part>
            <qtyOrd>50</qtyOrd>
        </purchaseOrderDet>
        <purchaseOrderDet>
            <operation>A</operation>
            <line>02</line>
            <dueDate>2002-06-01</dueDate>
            <part>10-15000</part>
            <qtyOrd>20</qtyOrd>
        </purchaseOrderDet>

    </purchaseOrder>
</maintainPurchaseOrder>
```

The following XML schema describes the syntax of the preceding sample. The `PurchaseOrderType` definition is used in the QDoc request and is stored in a separate file and included in the top-level schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This schema would be stored in file
    maintainPurchaseOrder-1_0.xsd -->
<schema
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:enc="http://www.w3.org/2002/12/soap-encoding"
    xmlns:qdoc="http://www.qad.com/qdoc/eb"
    xmlns:qcom="http://www.qad.com/qdoc/common"
    targetNamespace="http://www.qad.com/qdoc/eb"
    xml:lang="EN"
    >
    <!-- Include SOAP Encoding definitions,
        common MFG/PRO QDoc fields,
        purchase order definitions. -->
    <import namespace="http://www.w3.org/2002/12/soap-encoding"
        schemaLocation="soap-encoding.xsd"/>
    <include schemaLocation="qdocCommon-eB_99.xsd"/>
    <include schemaLocation="PurchaseOrderType-eB_1.xsd"/>
    <element name="maintainPurchaseOrder">
        <complexType>
            <attributeGroup ref="qcom:commonAttributes"/>
            <sequence>
               <element name="purchaseOrder"
                   type="qdoc:PurchaseOrderType"
                   minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
        </complexType>
    </element>
</schema>
```

The following XML schema fragment defines `PurchaseOrderType` and `PurchaseOrderDetType`.

```
    <!-- This schema would be stored in file
        PurchaseOrderType.xsd -->
    <!-- Include SOAP Encoding definitions,
        common MFG/PRO QDoc fields,
        transaction comment types. -->
    <import namespace="http://www.w3.org/2002/12/soap-encoding"
        schemaLocation="soap-encoding.xsd"/>
    <import namespace="http://www.qad.com/qdoc/common"
        schemaLocation="qdocCommon-qad_1.xsd"/>
    <include schemaLocation="qdocCommon-eB_99.xsd"/>
    <include schemaLocation="TransCommentType-eB_1.xsd"/>

    <!-- PurchaseOrderType extends the common ApiTempTableType
        in order to pick up the common temp-table fields -->
    <!-- Primary key(s) of PurchaseOrderType are listed in
        qdoc:primaryKeys for use when normalizing the data -->
    <complexType name="PurchaseOrderType" qdoc:primaryKeys="nbr">
        <complexContent>
           <extension base="qdoc:ApiTempTableType">
              <sequence>
                  <!-- Note: OperationType is defined in the common
                          QDoc XML Schema file to allow the
                          enumerated Values "A" (add), "M" (modify),
                          "R" (remove), and "S" (sync). -->
              <element name="operation"
                  type="qdoc:OperationType"
                  minOccurs="0"/>
              <element name="nbr" type="string" minOccurs="0"/>
              <element name="vend" type="string" minOccurs="0"/>
               ...
              <!-- Reuse common TransCommentType here -->
              <element name="purchaseOrderCmt"
                  type="qdoc:TransCommentType"
                  minOccurs="0" maxOccurs="unbounded"/>
              <element name="purchaseOrderDet"
                  type="qdoc:PurchaseOrderDetType"
                  minOccurs="0" maxOccurs="unbounded"/>
           </sequence>
        </extension>
    </complexContent>
</complexType>
...

<!-- PurchaseOrderDetType extends the common
    ApiTempTableType in order to pick up the common
    temp-table fields -->
<!-- Primary key(s) of PurchaseOrderDetType are listed in
    qdoc:primaryKeys for use when normalizing the data -->
<complexType name="PurchaseOrderDetType"
    qdoc:primaryKeys="line">
    <complexContent>
        <extension base="qdoc:ApiTempTableType">
           <sequence>
              <element name="operation" type="qdoc:OperationType"
                  minOccurs="0"/>
```

```
                <element name="line" type="int" minOccurs="0"/>
                ...
            </sequence>
        </extension>
    </complexContent>
</complexType>
...
```

The following illustrates the code API call that would be executed in order to process this sample QDoc, where all the application data is contained inside temp-tables.

```
run maintainPurchaseOrder in purchaseOrderAPI (
    input-output table ttPurchaseOrder,
    input-output table ttPurchaseOrderCmt,
    input-output table ttPurchaseOrderDet,
    input-output table ttPurchaseOrderDetCmt,
    input-output table ttImportExport,
    input-output table ttImportExportDet,
    input-output table ttTaxDet,
    input-output table ttScheduleCrossRef,
    input scopeTrans,
    input logTrans,
    input transId,
    input suppressResponseDetail,
    output table temp_err_msg,
    output success).
```

MFG/PRO QDoc Response

This is a sample partial QDoc response to the preceding QDoc request based on the API method `maintainPurchaseOrderResponse`. In this example, as with most code and JIT/S API methods, the input data is echoed back as output with any changed or default values, as well as any exception messages at the bottom.

See “QDoc Message Envelope” on page 221.

The QDoc message envelope is omitted, as it is described elsewhere. The examples presume that the earliest supported MFG/PRO release is eB.

```
<?xml version="1.0" encoding="UTF-8"?>
<maintainPurchaseOrderResponse version="eB_1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:enc="http://www.w3.org/2002/12/soap-encoding"
    xmlns:rpc="http://www.w3.org/2002/12/soap-rpc"
    xmlns="http://www.qad.com/qdoc/eb"
    xmlns:qcom="http://www.qad.com/qdoc/common"
    schemaLocation="http://www.qad.com/qdoc/eb/schemas"
    xml:lang="EN"
    >
    <rpc:result>returnValue</rpc:result>
    <returnValue>success</returnValue>
    <purchaseOrder>
        <nbr>PO1234</nbr>
        <vend>00000001</vend>
        <ordDate>2002-04-01</ordDate>
        <rmks>This is a test PO</rmks>
        ...
        <purchaseOrderCmt>
            <seq>01</seq>
            ...
            <cmmt>Line 1 of comment!</cmmt>
        </purchaseOrderCmt>
        <purchaseOrderDet>
            <line>01</line>
            <dueDate>2002-05-01</dueDate>
            <part>10-10000</part>
            <qtyOrd>50</qtyOrd>
            ...
        </purchaseOrderDet>
        <purchaseOrderDet>
            <line>02</line>
            <dueDate>2002-06-01</dueDate>
            <part>10-15000</part>
            <qtyOrd>20</qtyOrd>
            ...
        </purchaseOrderDet>
    </purchaseOrder
    <qcom:exception>
        <qcom:number>864</qcom:number>
        <qcom:description>SITE ADDRESS DOES NOT EXIST
        </qcom:description>
        <qcom:severity>error</qcom:severity>
        <qcom:field>site</qcom:field>
        <qcom:context>PO1234</qcom:context>
    </qcom:exception>
</maintainPurchaseOrderResponse>
```

The following XML schema fragment describes the syntax of the previous sample. The `PurchaseOrderResponseType` definition is used in the QDoc response and is stored in a separate file and included in the top-level schema. This defaults to include primary fields and can be replaced by a user-defined file that contains user-defined fields.

```
    <complexType name="PurchaseOrderResponseType">
        <sequence>
            <element name="nbr" type="string" minOccurs="0">
                <annotation>
                    <documentation>Purchase Order</documentation>
                </annotation>
            </element>
            <element name="vend" type="string" minOccurs="0">
                <annotation>
                    <documentation>Vendor</documentation>
                </annotation>
            </element>
            <element name="ordDate" type="string" minOccurs="0">
                <annotation>
                    <documentation>Order Date</documentation>
                </annotation>
        </element>
        <element name="rmks" type="string" minOccurs="0">
            <annotation>
                <documentation>Remarks</documentation>
            </annotation>
        </element>
        <element name="purchaseOrderCmt" type=
        "qdoc:PurchaseOrderCmtResponseType" minOccurs="0"
        maxOccurs="unbounded"/>
        <element name="purchaseOrderDet" type=
        "qdoc:PurchaseOrderDetResponseType" minOccurs="0"
        maxOccurs="unbounded"/>
    </sequence>
</complexType>
<complexType name="PurchaseOrderCmtResponseType">
    <sequence>
        <element name="seq" type="string" minOccurs="0">
            <annotation>
                <documentation>Sequence</documentation>
            </annotation>
        </element>
        <element name="cmmt" type="string" minOccurs="0">
            <annotation>
                <documentation>Trans Comment</documentation>
            </annotation>
        </element>
    </sequence>
</complexType>
<complexType name="PurchaseOrderDetResponseType">
    <sequence>
        <element name="line" type="integer" minOccurs="0">
            <annotation>
                <documentation> Ln</documentation>
            </annotation>
        </element>
        <element name="dueDate" type="integer" minOccurs="0">
            <annotation>
                <documentation> Due Date</documentation>
            </annotation>
        </element>
        <element name="part" type="integer" minOccurs="0">
            <annotation>
                <documentation> Part</documentation>
            </annotation>
        </element>
        <element name="qtyOrd" type="integer" minOccurs="0">
            <annotation>
                <documentation> Quantity Ordered</documentation>
            </annotation>
        </element>
    </sequence>
</complexType>
```

QDoc Message Envelope

Q/LinQ currently stores all control information regarding its export and import documents in non-XML data known as document control tags. While it is able to both export and import documents preceded by these tags, the tag format is non-XML and Q/LinQ-specific. These tags are not processed easily by XML-based tools such as XSLT stylesheets.

A standard, QAD-specific message envelope for use by QXtend as well as Q/LinQ is defined in this section using SOAP 1.2 syntax. It includes a minimal set of elements that must be understood by both Q/LinQ and QXtend, although additional product-specific elements can be added to it as needed in order to support interoperability with third parties. Its content is also loosely based on the envelopes/headers prescribed by the JMS, OAGIS, and ebXML specifications.

The present version of the QDoc envelope is likely to change significantly in future releases, as future XML-based messaging standards are published and adopted over time by the software industry.

While the QDoc message envelope is designed primarily to contain QDocs, it can also be used to contain any other type of document exported or imported by Q/LinQ such as OAGIS BODs. Its use is required for all QDocs.

Envelope Syntax

Based on SOAP standards, all QDoc messages consist of the XML root element `Envelope` with two child elements, `Header` and `Body`. The `Body` element contains a single QDoc as described in the preceding sections as its only payload. The `Header` element consists of one or more header block elements named `qdoc` as described in the following section.

The `Envelope`, `Header`, and `Body` elements are defined and must be labeled using the following SOAP Envelope namespace URI:

http://www.w3.org/2002/12/26/soap-envelope

They may contain attributes, but these may be ignored.

Header Block Content

Each QDoc header block consists of a root element QDoc containing the elements described in this section. Header blocks can also contain any of the `role`, `mustUnderstand`, and `encodingStyle` attributes, as defined by the SOAP 1.2 specification.

- The `role` attribute may have any legal URI as its value.
- The `mustUnderstand` attribute is boolean.

Both are ignored on inbound QDocs.

The `encodingStyle` attribute is a URI, and is set to the following value on all outbound QDoc header blocks:

http://www.w3.org/2002/12/26/soap-encoding

While more than one header block inside a single message is syntactically allowed, in the case of inbound QDocs all QDoc headers after the first one are ignored. All outbound QDocs contain only a single QDoc header block.

Requestor Element

The requestor element can contain requestor type, user ID, and password attributes. For licensing only, the requestor type contains the licensed requestor.

```
<requestor type=“mfgpro” />
```

This is only necessary if the user has anything other than an unlimited license.

For security and traceability, if the `useQDocRequestor` attribute is set to `true` in *`TOMCAT_HOME`*`/webapps/qxtendserver/WEB-INF/conf/qxtendconfig.xml`, the user ID and password for connection to the requestor (for example, MFG/PRO), must be included in the requestor element.

```
<requestor id=“user” password=”password”/>
```

This password is sent in text form, encrypted within the server, and is encrypted in any response QDocs.

The requestor element can also contain all three attributes as required.

```
<requestor id="user" password="password" type="mfgpro" />
```

The requestor element can also be left out of the header altogether if the QXtend implementation does not require user ID and password, and has an unlimited license.

senderId

The `senderId` element identifies the sender of the QDoc or other document. It is defined as a URI so that future QAD product releases can more easily use Internet-based industry standards for organizational identification. Existing B2B standards such as EDIRA (ISO 6523), EDIFACT ISO 9735, or ANSI ASC X12 105 registries may be the basis for such identification schemes.

While most of the URI is not yet used, it must include the query parameter `sender` in order to identify the sending entity within a given installation. The syntax of the query parameters is similar to that commonly used with any URI, as in the following example:

http://anyBaseURI/path1/path2?sender=myCompany&amp;…

The ampersand character separating the query parameters must be written as `&amp;` in order to pass XML well-formedness parsing rules.

In the case of outbound documents sent by Q/LinQ, sender is always equal to the Q/LinQ System ID (@SYSID tag) maintained in the Q/LinQ System Control table. This tag must have the same value as the configuration parameter used by the QDoc server to identify a given MFG/PRO database in a multi-database installation.

In the case of inbound documents, sender is stored in the Q/LinQ trading partner ID (@TRADPTRID tag) field. The trading partner ID identifies the external owner of the MFG/PRO object about which Q/LinQ is being notified.

receiverId

The `receiverId` element identifies the receiver of the QDoc or other document. It is defined as a URI so that future QAD product releases can more easily use Internet-based industry standards for organizational identification. Existing B2B standards such as EDIRA (ISO 6523), EDIFACT ISO 9735, or ANSI ASC X12 105 registries may be the basis for such identification schemes.

While most of the URI is not yet used, it must include the query parameters `connection` and `receiver` in order to identify the receiving entity within a given installation. The syntax of the query parameters is similar to that commonly used with any URI, as in the following example:

> http://anyBaseURI/path1/path2?connection=
> yourQueue&amp;receiver=yourCompany&amp;…

In the case of outbound documents sent by Q/LinQ, receiver is the Q/LinQ trading partner ID (@TRADPTRID tag) for which the export document was published. In many cases, this value corresponds to the external owner of the MFG/PRO data object that is the main subject of the document. connection is the Q/LinQ application ID (@APPID tag) through which the document was sent.

In the case of Q/LinQ inbound documents, receiver must be set to the Q/LinQ System ID (@SYSID tag) field maintained in the Q/LinQ System Control table, which in turn must be the same as the identifier used in the QDoc server to identify the target MFG/PRO database. However, it is ignored by the current version of Q/LinQ, as each Q/LinQ installation is associated with one and only one MFG/PRO database and cannot properly route documents to any other MFG/PRO database. connection is defined by the sender, but is mapped by Q/LinQ to the Q/LinQ application ID (@APPID tag) through which the document is received.

senderDocumentId

The `senderDocumentId` element is the sender's locally unique identifier for the QDoc, meaningful to the sender's messaging software but not necessarily to the source application. It is required and must be referenced in the `originalDocumentRef` field on QDoc confirmations so that the receiver of the confirmation—the sender of the original document—can unambiguously identify the document being confirmed.

The `senderDocumentId` element is Q/LinQ's internally assigned document ID. It is not significant outside of Q/LinQ, but it provides a potentially useful cross-reference back to the Q/LinQ database.

documentStandard

The documentStandard element is intended to identify the standard or specification, if any, to which the document conforms. This could be an externally published standard such as EDIFACT, OAGIS, ANSI X12, or UCCnet. It could also be an internal identifier for a proprietary grammar used at one particular installation, for those QAD application users who have defined standard business objects for use inside their enterprise.

The special value QDoc is reserved by QAD to designate QDocs. documentStandard is an optional element.

This element is exactly equivalent to the Q/LinQ document standard or @DOCSTD document control tag. As with Q/LinQ, the special value cim is reserved by QAD to designate documents that can be processed by the MFG/PRO CIM Interface or the UI-emulation mode of Q/LinQ.

documentType

The documentType element identifies the type of document contained in the message payload (that is, the SOAP Body), and is used to route inbound documents to the appropriate handler for processing. It is semantically equivalent to a message type, action, or service designator. It is an optional element, and may be assigned a default value by the QAD Integration Framework based on the needs of the installation.

For QDocs, this element should be set to the name of the QDoc; for example, maintainSalesOrder.

This element is exactly equivalent to the Q/LinQ document type or @DOCTYP document control tag. For documents with a documentStandard of cim, it should be set to the name of the MFG/PRO program called to process the document.

documentVersion

The documentVersion element qualifies the documentType with a version or revision designator that permits different variations of the same type of document to be distinguished for special handling. For documents conforming to an externally published standard, such as EDI, it is set to the version or revision level designator assigned by the published standard.

If documentVersion is omitted, it is presumed to designate the most current commercial or public version of the associated document type. When used with internal, less formal document grammars that do not use any form of version control, it would be omitted.

For QDocs, this element should be set to the version attribute of the QDoc; for example, eB_1.

This element is exactly equivalent to the Q/LinQ document revision or @DOCREV document control tag.

confirmationLevel

The confirmationLevel element designates under what circumstances a QDoc Confirmation document is requested. Allowed values are none, error, or all.

This element has exactly the same meaning as the present Q/LinQ tag @ACKLVLREQD.

dateTimeCreated

The dateTimeCreated element contains the date- and time-stamp for the creation of the document, expressed using the DateTime type prescribed by XML schema. For example, the following value would apply to a document created on July 1, 2002, at 9:00am Pacific Standard Time:

```
2002-07-01T09:00:00-0800
```

This element has exactly the same meaning as the present Q/LinQ tags @DATECREATE, @TIMECREATE, and @TIMEZONE in combination.

senderDocumentRef

The senderDocumentRef is a flexible-format, multi-occurrence string providing a reference back to the sending application. It is not necessarily unique and serves only to provide a meaningful reference for application end users, as opposed to the `senderDocumentId` used only for document matching purposes. Accordingly, it should be assigned to one or more values that represent the key fields of the application data objects most closely associated with the document.

For example, in the case of a sales order document it might contain the sales order number. In the case of inventory balances, it might contain the SKU number of the material as well as the identifier of the warehouse/facility in which the material is stored.

Q/LinQ populates only two occurrences of this field per document at most. If the corresponding MFG/PRO data object has a primary site code associated with it, as in the case of inventory, the first occurrence is set to the MFG/PRO site (@MFGPROSITE) tag and the second occurrence to the MFG/PRO key (@MFGPROKEY) tag. Otherwise, the first occurrence is set to the MFG/PRO key (@MFGPROKEY) tag.

receiverDocumentRef

The `receiverDocumentRef` is a flexible-format, multi-occurrence string providing a reference back to the receiving application. It is not necessarily unique and serves only to provide a meaningful reference for application end users. It is analogous to the `senderDocumentRef` field, and should be assigned to one or more values that represent the key fields of the application data objects most closely associated with the document.

In most cases, this field is omitted from the message envelope, as the sender application or messaging agent would not normally have a meaningful document reference expressed in terms of the receiver. However, it may be useful for follow-up messages in which two applications are engaged in a long-running dialog about data objects known to both. For example, MFG/PRO might use this field on a shipment notification to reference an external sales order number and line.

Q/LinQ expects only two occurrences of this field per document at most:

- If the corresponding MFG/PRO data object has a primary site code associated with it, as in the case of inventory, the first occurrence is assumed to be a site code and is stored in the MFG/PRO site (@MFGPROSITE) tag.

- The second occurrence is assumed to be some other key field and is stored in the MFG/PRO key (@MFGPROKEY) tag. Otherwise, the first occurrence is stored in the MFG/PRO key (@MFGPROKEY) tag.

SOAP Compliance Limitations

The QDoc message envelope implements the SOAP 1.2 specification with the following limitations and qualifications:

- The QDoc solution supports document-style message exchange as opposed to SOAP RPC. In order to make support for SOAP RPC and related standards such as WSDL easier to implement in future releases, SOAP encoding rules are used to represent all QDoc data as described in earlier sections of this chapter.
- All QDoc messages require a `Header` element, whereas SOAP considers this element optional.
- QAD applications ignore the `role` attribute on all SOAP header blocks, acting as the ultimate SOAP receiver in all cases. In other words, a QAD application automatically plays all roles that appear in the SOAP message.
- QAD applications ignore the `mustUnderstand` attribute on all SOAP header blocks, assuming that it must understand and process every received QDoc. In other words, they always act as if `mustUnderstand` is set to true.
- QAD applications cannot act as SOAP intermediaries, passing the QDoc to other SOAP nodes for processing. Rather, they process all inbound messages as the ultimate receiver.
- QAD applications process a received QDoc only once, regardless of the number of SOAP header blocks that may be included inside the `Header` element of the envelope.

Future Extensions and Forward Compatibility

The present content of the QDoc envelope, while mostly SOAP compliant, is entirely QAD defined with no reference to external industry standards or specifications outside of SOAP. As XML- and SOAP-based messaging standards mature in the marketplace, the QDoc envelope will require additions and changes to address authentication, privacy, reliability, and work flow requirements among others.

QDoc Envelope Examples

The following example illustrates a QDoc envelope.

Envelope Example

```
<!-- Here is an inbound QDoc request message envelope. -->
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <env:Header>
        <!-- SOAP 'mustUnderstand' and 'role' attributes areignored
        but may be present -->
        <QDoc xmlns="http://www.qad.com/qdoc/common"
        env:mustUnderstand="true" env:role="http://www.w3.org/2002/
        12/soap-envelope/role/next">
            <senderId>http://www.acme.com/site1?sender=site1</
            senderId>
            <receiverId>http://www.acme.com/site2?connection=
            queue1&amp;receiver=db2</receiverId>
            <senderDocumentId>eq200000</senderDocumentId>
            <requestor id="user" password="apasswd" type="mfgpro"/>
            <!-- descriptor may be omitted if Q/LinQ is set up to
            expect QDocs -->
            <documentStandard>QDoc</documentStandard>
            <documentType> maintainSalesOrder</documentType>
            <documentVersion/>
            <confirmationLevel>all</confirmationLevel>
            <dateTimeCreated>2002-07-10T09:00:00-0800</
            dateTimeCreated>
            <senderDocumentRef>eqso5678</senderDocumentRef>
            <senderDocumentRef>1</senderDocumentRef>
        </QDoc>
    </env:Header>
    <env:Body>
        <!-- The QDoc goes here -->
        <maintainSalesOrder …>
                ...
        </maintainSalesOrder>
    </env:Body>
</env:Envelope>
```

XML Schema for Envelope Header

Following is the unannotated XML schema describing the syntax of the header block of the QDoc document envelope. It makes no distinction between export and import, so that exported Q/LinQ documents can be reprocessed as import documents with no XML syntax errors. The elements and attributes are defined in an envelope-specific namespace rather than the primary product-specific QDoc namespaces.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.qad.com/qdoc/common" xmlns:qcom=
"http://www.qad.com/qdoc/common" xmlns="http://www.w3.org/2001/
XMLSchema" xmlns:soapenv="http://www.w3.org/2002/12/soap-envelope"
xmlns:enc="http://www.w3.org/2002/12/soap-encoding"
elementFormDefault="qualified" xml:lang="EN">
    <import namespace="http://www.w3.org/2002/12/soap-envelope"
    SchemaLocation="soap-envelope.xsd"/>
    <complexType name="QdocEnvelopeType">
        <sequence>
            <element name="senderId" type="anyURI"/>
            <element name="senderRef" type="string" minOccurs="0"/>
            <element name="receiverId" type="anyURI"/>
            <element name="receiverRef" type="string" minOccurs="0"/>
            <element name="senderDocumentId" type="string"/>
            <element name="documentStandard" type="string"/>
            <element name="documentType" type="string"/>
            <element name="documentVersion" type="string"/>
            <element name="confirmationLevel" default="none"
            minOccurs="0">
                <simpleType>
                    <restriction base="string">
                        <enumeration value="none"/>
                        <enumeration value="error"/>
                        <enumeration value="all"/>
                    </restriction>
                </simpleType>
            </element>
            <element name="dateTimeCreated" type="dateTime"
            minOccurs="0"/>
            <element name="senderDocumentRef" type="string"
            minOccurs="0" maxOccurs="unbounded"/>
            <element name="receiverDocumentRef" type="string"
            minOccurs="0" maxOccurs="unbounded"/>
            <element name="requestor" minOccurs="0" maxOccurs=
            "unbounded">
                <complexType>
                    <attributeGroup ref="qcom:requestorAttributes"/>
                </complexType>
            </element>
            <any minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
        <attribute ref="soapenv:encodingStyle"/>
        <attribute ref="soapenv:role"/>
        <attribute ref="soapenv:mustUnderstand"/>
        <anyAttribute/>
    </complexType>
    <element name="qdoc" type="qcom:QdocEnvelopeType"/>
    <attributeGroup name="requestorAttributes">
        <attribute name="id" type="string" use="optional"/>
        <attribute name="type" type="string" use="optional"/>
        <attribute name="password" type="string" use="optional"/>
    </attributeGroup>
    <attributeGroup name="commonAttributes">
        <attribute name="version" type="string" use="required"/>
        <attribute name="scopeTransaction" type="boolean" use=
        "optional"/>
        <attribute name="logTransaction" type="boolean" use=
    "optional"/>
    <attribute name="transactionId" type="string" use=
    "optional"/>
    <attribute name="suppressResponseDetail" type="boolean" use=
    "optional"/>
    <attribute ref="xml:lang"/>
</attributeGroup>
<attributeGroup name="arrayEntryAttributes">
    <attribute name="index" type="positiveInteger" use=
    "required"/>
    <attribute name="skip" type="boolean"/>
</attributeGroup>
<complexType name="EntryString">
    <simpleContent>
        <extension base="string">
            <attributeGroup ref="qcom:arrayEntryAttributes"/>
        </extension>
    </simpleContent>
</complexType>
<complexType name="EntryInteger">
    <simpleContent>
        <extension base="integer">
            <attributeGroup ref="qcom:arrayEntryAttributes"/>
        </extension>
    </simpleContent>
</complexType>
<complexType name="EntryDecimal">
    <simpleContent>
        <extension base="decimal">
            <attributeGroup ref="qcom:arrayEntryAttributes"/>
        </extension>
    </simpleContent>
</complexType>
<complexType name="EntryDate">
    <simpleContent>
        <extension base="date">
            <attributeGroup ref="qcom:arrayEntryAttributes"/>
        </extension>
    </simpleContent>
</complexType>
<complexType name="ExceptionType">
    <sequence>
        <element name="number" type="string"/>
        <element name="description" type="string"/>
        <element name="severity" default="error" minOccurs="0">
            <simpleType>
                <restriction base="string">
                    <enumeration value="informational"/>
                    <enumeration value="warning"/>
                    <enumeration value="error"/>
                </restriction>
            </simpleType>
        </element>
        <element name="field" type="string" minOccurs="0"/>
        <element name="context" type="string" minOccurs="0"/>
        <element name="trace" type="string" minOccurs="0"/>
    </sequence>
</complexType>
```

```
    <simpleType name="ReturnValueType">
        <restriction base="string">
            <enumeration value="success"/>
            <enumeration value="warning"/>
            <enumeration value="error"/>
        </restriction>
    </simpleType>
    <complexType name="QdocConfirmationType">
        <sequence>
            <element name="originalDocumentRef" type=
            "qcom:QdocEnvelopeType"/>
            <element name="processingStage">
                <simpleType>
                    <restriction base="string">
                        <enumeration value="received"/>
                        <enumeration value="accepted"/>
                        <enumeration value="processed"/>
                    </restriction>
                </simpleType>
            </element>
            <element name="processingOutcome">
                <simpleType>
                    <restriction base="string">
                        <enumeration value="error"/>
                        <enumeration value="warning"/>
                        <enumeration value="success"/>
                    </restriction>
                </simpleType>
            </element>
            <element name="exception" type="qcom:ExceptionType"
            minOccurs="0" maxOccurs="unbounded"/>
            <element name="processingResults" minOccurs="0">
                <complexType>
                    <sequence>
                        <any processContents="lax" minOccurs="0"
                        maxOccurs="unbounded"/>
                    </sequence>
                </complexType>
            </element>
        </sequence>
        <attributeGroup ref="qcom:commonAttributes"/>
    </complexType>
</schema>
```

SOAP Faults

As previously discussed, any errors detected in the preprocessing of an inbound QDoc are thrown to the sender as SOAP faults. SOAP faults are SOAP response messages that contain a single instance of the `Fault` element inside the SOAP body, with no other data content.

The `Fault` element in turn contains:

- Mandatory `Code` element
- Mandatory `Reason` element
- Optional `Node`, `Role`, and `Detail` elements

The purpose of SOAP faults is to describe a single fatal error to the sender of the original SOAP message using a standard syntax.

The `Code` element values for common errors, as well as related `Subcode` element values, are predefined in the SOAP 1.2 specification:

- `VersionMismatch`: The message envelope does not conform to the SOAP specification.
- `MustUnderstand`: A `Header` element was not understood even though the SOAP `mustUnderstand` attribute was set to true. This is not applicable to QDocs, as the `mustUnderstand` attribute is not used.
- `DataEncodingUnknown`: The SOAP `encodingStyle` attribute has a value other than:

  http://www.w3.org/2002/12/soap-encoding
- `Sender`: The QDoc was incorrectly formed by the sender in some fashion not covered by the other general error conditions, as further described by the `Subcode` element.
- `Receiver`: The QDoc could not be processed by the receiver for reasons other than its syntax or content, as further described by the `Subcode` element.

The following `Subcode` value elements are used with the `Sender` code to describe errors in the QDoc request syntax:

- `QDocNotWellFormed`: The QDoc body's XML content is not well formed and cannot be parsed.
- `QDocEnvelopeInvalid`: The QDoc's envelope does not have valid syntax, and the contained QDoc cannot be unwrapped.
- `QDocNotRecognized`: No QDoc with the given name is supported by the installation.

- `QDocSyntaxInvalid`: The QDoc is recognized and its XML content is well formed, but it does not have valid syntax with respect to its XML schema. For example, the wrong input parameters are provided.
- `QDocReceiverNotRecognized`: The `receiver` attribute of the QDoc message envelope was not recognized; therefore, the QDoc cannot be processed.
- `QDocVersionNotSupported`: No QDoc is supported by the installation with the given combination of `name`, `version`, and `receiver` attributes.

The following `Subcode` value element is used with the `Receiver` code to describe errors in the QAD application responsible for processing the QDoc request:

- `QDocApplicationNotAvailable`: The QAD application responsible for processing the QDoc request is not available. For example, the MFG/PRO database server is not running, or the QDoc server is not able to communicate with Progress clients or AppServers.

The `Detail` element of a QDoc SOAP `Fault` contains one or more `exception` elements of the type `ExceptionType` describing the nature of the preprocessing problem. In all such exceptions, the number element is blank, severity is error, and description contains an error message describing the problem. The possible error messages consist of text equivalent to the QDoc-specific `Subcode` values listed here.

The SOAP `Reason` element is set to the value of the description element of exception, essentially the text of the error message that would be displayed to a human user. The SOAP `Node` and `Role` elements are not needed or used, as it is presumed that the QAD application is always the ultimate receiver of the SOAP message.

The following XML schema fragment defines the QAD-specific Subcodes.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
           xmlns:qcom="http://www.qad.com/qdoc/common"
           targetNamespace="http://www.qad.com/qdoc/common">
    <element name="soapFaultSubcodeValue">
        <simpleType>
            <restriction base="QName">
               <enumeration value="qcom:qdocNotWellFormed"/>
               <enumeration value="qcom:qdocEnvelopeInvalid"/>
               <enumeration value="qcom:qdocNotRecognized"/>
               <enumeration value="qcom:qdocSyntaxInvalid"/>
               <enumeration value="qcom:qdocReceiverNotRecognized"/>
               <enumeration value="qcom:qdocVersionNotSupported"/>
               <enumeration
                   value="qcom:qdocApplicationNotAvailable"/>
            </restriction>
        </simpleType>
    </element>
</schema>
```

This partial example of an outbound MFG/PRO QDoc response contains a SOAP fault.

```
<!-- Here is an outbound QDoc response message envelope
    containing a SOAP fault. -->
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
    xmlns:env="http://www.w3.org/2002/12/soap-envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <env:Header>
        ...
    </env:Header>
    <env:Body>
        <!-- The SOAP fault follows, with no other QDoc content
            allowed -->
        <env:Fault>
            <env:Code>
               <env:Value>env:Sender</env:Value>
               <env:Subcode>
                   <!-- Subcode value is QAD-defined -->
                   <env:Value
xmlns:qcom="http://www.qad.com/qdoc/common">
                       QDocNotWellFormed
                   </env:Value>
               </env:Subcode>
            </env:Code>
            <!-- Reason is taken from the QDoc exception description
               -->
            <env:Reason xml:lang="en-US">
               QDoc is not well formed
            </env:Reason>
            <env:Detail>
               <!-- Here is the QDoc exception element -->
               <qcom:exception
                   xmlns:qcom="http://www.qad.com/qdoc/common">
                   <qcom:number></qcom:number>
                   <qcom:description>
                       QDoc is not well formed
                   </qcom:description>
                   <qcom:severity>error</qcom:severity>
                   <!-- field and context are not relevant to this
                       type of error -->
                   <qcom:field/>
               <qcom:context/>
            </qcom:exception>
            </env:Detail>
            ...
        </env:Fault>
    </env:Body>
</env:Envelope>
```

QDoc Processing Confirmation

Closely associated with the QDoc message envelope is another general-purpose feature required in order to support asynchronous document processing: the QDoc confirmation document. This QDoc provides a delayed response to an earlier QDoc request that was processed asynchronously.

Many integration scenarios—both A2A and B2B—need to process inbound requests asynchronously with a delayed, generic response. In this case, the request is almost always a business document being submitted for processing, such as an order or data object to be updated, rather than a query-oriented request. The response must typically include feedback regarding the status of the request, sometimes only on an exception basis; but not necessarily business data resulting from the request as would be present in a normal QDoc response document.

Several industry consortia, notably OAG and ebXML, have specified or identified such general-purpose confirmations as critical components of application interoperability. Also, traditional EDI standards have long included general-purpose process acknowledgment transactions. The QDoc confirmation is QAD’s native implementation of the same concept.

Confirmation Content

The QDoc confirmation document is based on the OAGIS Confirm BOD, modified syntactically to reuse other QDoc elements previously defined. Each QDoc confirmation consists of a root element `qdocConfirmation` containing the elements described in this section. It applies to all QAD products, and is, therefore, defined inside the QAD common rather than MFG/PRO-specific namespace URI.

Original Document Reference

The originalDocumentRef element is essentially a copy of the QDoc envelope of the original QDoc request. When used as a response for non-QDoc requests, it provides the available data about the request that would logically have been contained in the QDoc envelope if the request had been a QDoc. It contains the same elements as the `QDocEnvelope`, and is syntactically defined as an instance of the same XML schema type:

- `senderId`
- `receiverId`
- `senderDocumentId`
- `documentStandard`
- `documentType`
- `documentVersion`
- `confirmationLevel`
- `dateTimeCreated`
- `senderDocumentRef`
- `receiverDocumentRef`

When sent within a QDoc message envelope, the QDoc confirmation contains envelope information for two QDocs: itself and the earlier request to which it is responding. The receiving application may have to use information from both envelopes in order to process the confirmation properly. For example, the envelope for the original request may include information necessary for the source system to cross-reference the confirmation to the request; but the confirmation's envelope may include current cross-references to local application data (for example, MFG/PRO site or key values) that are more relevant to application end-users.

Processing Stage

The `processingStage` element describes how far the receiving application has attempted to process the earlier QDoc request. It is used in combination with the `processingOutcome` element to completely describe the present status of the request. This element is based on the processing stage field of Q/LinQ.

There are three possible values:

- received: The request was received successfully, but has not yet been evaluated for syntactic or semantic correctness. Thus, it may be valid or invalid.
- accepted: The request was received, parsed, and preprocessed far enough to determine that is valid. However, it has not yet been processed by the receiving application and, in a B2B scenario, may or may not be accepted from a business point of view. This value is roughly equivalent to the mapped processing stage of Q/LinQ.
- processed: The request has been received and processed into the application.

While confirmations in most interoperability scenarios are required only after application processing, ebXML defines the potential need for an acceptance acknowledgment as well as a postprocessing response document. Therefore, the accepted stage is included for the sake of completeness and future extensions to the QDoc solution.

Processing Outcome

The `processingOutcome` element describes how far the receiving application has attempted to process the earlier QDoc request. It is used in combination with the `processingStage` element to completely describe the present status of the request. This element is based on the error status field of Q/LinQ.

There are three possible values.

- success: The request was successfully processed through the associated processing stage.
- warning: The request was successfully processed through the associated processing stage. However, one or more warning-level exceptions were raised.

- error: The request caused one or more errors to be raised and was not successfully processed. The errors may have been caused by either invalid request content/format or errors in the receiving application such as incorrect configuration settings or missing application data.

Exception Messages

In case errors or warnings were raised during receiving, preprocessing, or application processing, one or more instances of the exception element are provided describing the exception conditions with human-readable diagnostic messages. This is the same XML schema type as the exception element present in other QDoc responses.

Processing Results

The `processingResults` element contains the results of processing the original QDoc, at least for those inbound QDocs to which such detail is relevant. In future releases of QXtend, it may be the body of the QDoc response that would have been returned had the inbound QDoc been processed as a synchronous request. Its syntax and content are entirely dependent on the type of the earlier QDoc being confirmed.

Whether or not the `processingResults` element is present depends on whether the `suppressResponseDetail` attribute is set to true in the inbound QDoc. If true, `processingResults` is absent; otherwise, it is present.

Confirmation Examples

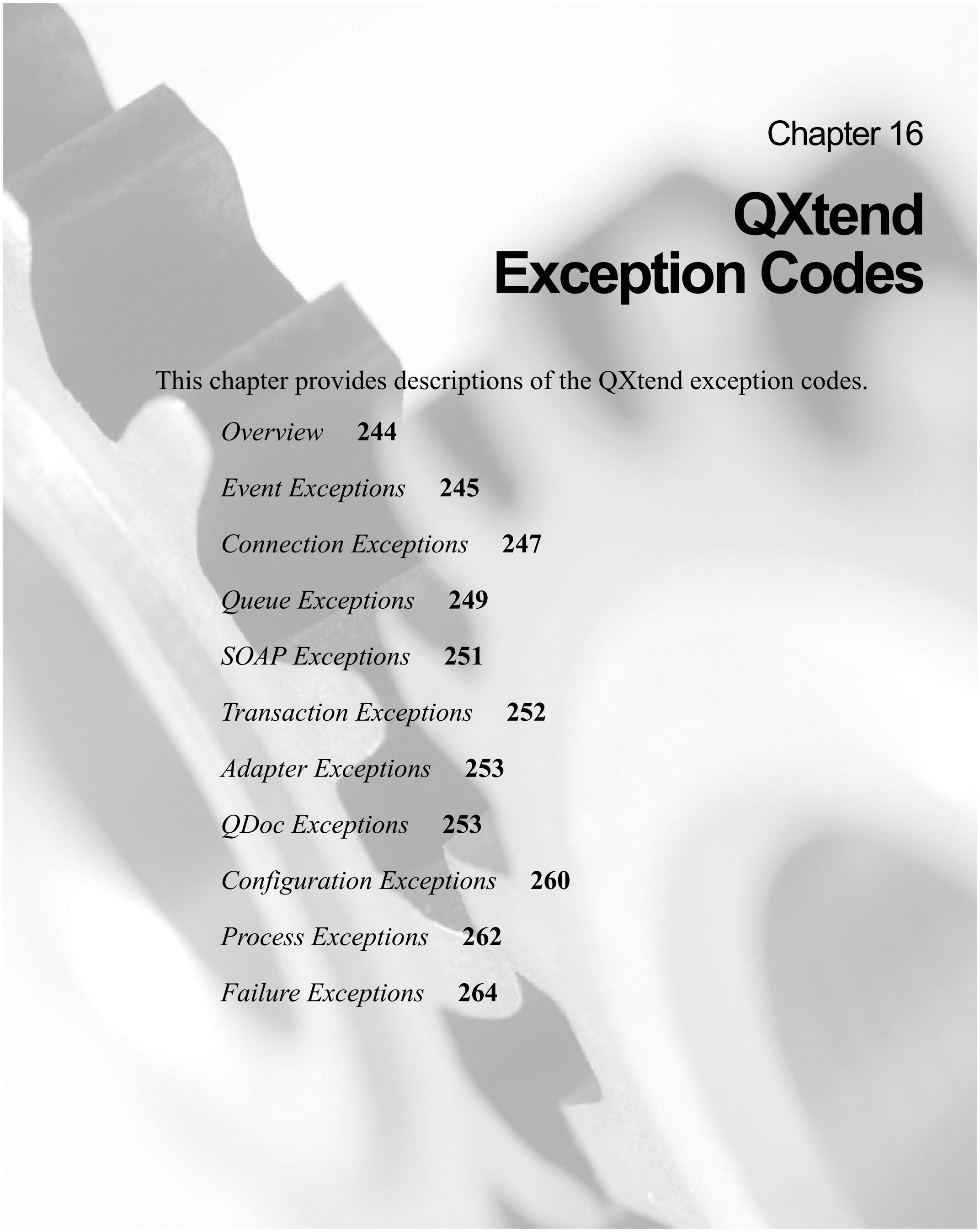
The following sample QDoc confirmation fragment was generated as a result of an MFG/PRO sales order originating in an external application. The example does not show the message envelope that could contain the confirmation document.

```
<?xml version="1.0" ?>
<qcom:qdocConfirmation version="eB_1.0"
    xmlns:qcom="http://www.qad.com/qdoc/common"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:enc="http://www.w3.org/2002/12/soap-encoding"
    schemaLocation="http://www.qad.com/qdoc/eb/schemas"
    xml:lang="EN"
    >
    <qcom:originalDocumentRef>
```

```
        <qcom:senderId>
http://www.acme.com/site1?sender=db2
        </qcom:senderId>
        <qcom:receiverId>
http://www.acme.com/site2?connection=queue2&amp;receiver=site1
        </qcom:receiverId>
        <qcom:senderDocumentId>00001234</qcom:senderDocumentId>
        <qcom:documentStandard>QDoc</qcom:documentStandard>
        <qcom:documentType>maintainSalesOrder</qcom:documentType>
        <qcom:documentVersion>eB_1</qcom:documentVersion>
        <qcom:confirmationLevel>all</qcom:confirmationLevel>
        <qcom:dateTimeCreated>2002-07-11T09:00:00-0800
        </qcom:dateTimeCreated>
        <qcom:senderDocumentRef>10000</qcom:senderDocumentRef>
        <qcom:senderDocumentRef>so12345</qcom:senderDocumentRef>
        <qcom:receiverDocumentRef>
            eqso5678
        </qcom:receiverDocumentRef>
    </qcom:originalDocumentRef>
    <qcom:processingStage>processed</qcom:processingStage>
    <qcom:processingOutcome>error</qcom:processingOutcome>
    <qcom:exception>
        <qcom:number>124</qcom:number>
        <qcom:description>
            TRAILER CODE NOT DEFINED
        </qcom:description>
        <qcom:severity>error</qcom:severity>
        <qcom:field>trl2Cd</qcom:field>
        <qcom:context>so12345</qcom:context>
    </qcom:exception>
    <qcom:processingResults>
        <qdoc:maintainSalesOrderResponse>
            ...
        </qdoc:maintainSalesOrderResponse>
    </qcom:processingResults>
</qcom:qdocConfirmation>
```

The following represents the unannotated XML schema for the preceding QDoc.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:enc="http://www.w3.org/2002/12/soap-encoding"
    xmlns:qcom="http://www.qad.com/qdoc/common"
    targetNamespace="http://www.qad.com/qdoc/common"
    xml:lang="EN"
    >
    <!-- Include SOAP Encoding definitions,
        common QDoc envelope fields. -->
    <import namespace="http://www.w3.org/2002/12/soap-encoding"
        schemaLocation="soap-encoding.xsd"/>
    <include schemaLocation="qdocCommon-qad_1.xsd"/>

    <element name="qdocConfirmation">
        <complexType>
            <sequence>
               <attributeGroup ref="qcom:commonAttributes"/>
               <element name="originalDocumentRef"
                   type="qcom:qdocEnvelopeType"/>
               <element name="processingStage">
                   <simpleType>
                       <restriction base="string">
                           <enumeration value="received"/>
                           <enumeration value="accepted"/>
                           <enumeration value="processed"/>
                       </restriction>
                   </simpleType>
               </element>
               <element name="processingOutcome">
                   <simpleType>
                       <restriction base="string">
                           <enumeration value="error"/>
                           <enumeration value="warning"/>
                           <enumeration value="success"/>
                       </restriction>
                   </simpleType>
               </element>
            <element ref="qcom:exception"
               minOccurs="0" maxOccurs="unbounded"/>
            <element ref="processingResults"
               minOccurs="0" maxOccurs="unbounded">
               <complexType>
                   <sequence>
                       <any minOccurs="0" maxOccurs="unbounded"
                           processContents="lax"/>
                   </sequence>
               </complexType>
            </element>
        </sequence>
    </complexType>
</element>

</schema>
```



Chapter 16

QXtend Exception Codes

This chapter provides descriptions of the QXtend exception codes.

Overview

See “Include Java Trace Information” on page 50 for information on the excludeTraces setting.

This chapter contains a complete list of possible QXtend Inbound exceptions. Exceptions can include other exceptions. An included exception is contained in the response QDoc (unless `excludeTraces` is set) and in the log files (regardless of `excludeTraces`); the included exception can provide more insight into the problem.

Table 16.1 provides the directory locations of the log and configuration files referenced in the exception codes. See “Logging” on page 53 for a discussion of all the QXI logging options.

Table 16.1 Configuration and Log Files by Location

Path	**Files**
TOMCAT_HOME\webapps\qxtendserver\qxtendQueues	directoryloaderconfig.xml soapconfig.xml
TOMCAT_HOME\webapps\qxtendserver\qxtendQueues\QUEUE_NAME	queueconfig.xml
TOMCAT_HOME\webapps\qxtendserver\WEB-INF\conf	config-files.xml connectionManagerConfig.xml environmentmanager.xml qxtendconfig.xml routings.xml
TOMCAT_HOME\webapps\qxtendserver\WEB-INF\descriptors\MFGPRO_VERSION	implementationQdocs.xml qadQdocs.xml (global)
TOMCAT_HOME\webapps\qxtendserver\WEB-INF\logs	connectionPools.log qdocInfo.log queue.log.*date* qxtendServer.log
TOMCAT_HOME\webapps\qxtendserver\WEB-INF\receivers	qdocReceivers.xml
TOMCAT_HOME\webapps\qxtendserver\WEB-INF\receivers\RECEIVER_NAME\descriptors	implementationQDocs.xml qadQdocs.xml (receiver specific)

Event Exceptions

Events are responsible for assisting in the navigation of the MFG/PRO screens. These exceptions may occur during the processing of a QDoc. The event exceptions are visible in the response QDoc and in the logs.

Table 16.2
Event Exceptions

Exception Code	**Description**	**Notes**
EventException001	Error processing data	During the processing of a QDoc request, QXtend passed through the same field twice that was not the start of an iteration. The field in question can be found in the exception detail.
EventException002	Unexpected prompt field after delete	Unexpected field name in the events file for the delete confirmation prompt. The expected field name is usually the response to the MFG/PRO confirmation query: Are you sure you wish to delete this record? Review the events file or use QGen to update the file.
EventException003	Error while deleting	An MFG/PRO error occurred while deleting a record. Review the data in the QDoc and ensure that this data can be deleted.
EventException004	Unable to load an events document	The specified events file cannot be found or loaded. There may be invalid XML in the file.
EventException005	Error creating an event object	While adding a new events type, no Java class exists with the event name. Check the events file for the event types. This occurs only if the events file has been incorrectly updated. Review the events file for an invalid event type such as IterationEvent: `<IterationEvent iterationname= "supplier" exititeration="f4"/>`
EventException006	Object is not an event	The event object created (see EventException005) does not conform to a specific Java interface. This occurs only if the events file has been incorrectly updated. Review the events file for an invalid event type such as IterationEvent.

Exception Code	Description	Notes
EventException008	No name found for the iteration	Current field at the start of each iteration cannot be located in the events file. Review the events file for this program; it may need remapping using QGen.
EventException009	Invalid selection in selection list	The value entered in the QDoc for a selection list is not in MFG/PRO. Review the data in the QDoc and ensure that the selection item exists in MFG/PRO.
EventException010	No data in the QDoc	No data to process in the QDoc.
EventException011	Data sent too large	The data submitted for a primary field is too large. It may update an existing record rather than create a new record. Review the data in the QDoc.
EventException012	Ambiguous selection in selection list	Data for a selection list in the QDoc exists in more than one row in MFG/PRO. Review the data in the QDoc and MFG/PRO for this selection list.
EventException014	Preprocess program name is null	The program name for the preprocess event is null. Verify the name in the events file.
EventException015	Error calling preprocess program	An error occurs while calling the preprocessor program. Review the events file.
EventException016	Preprocess – Updated iteration node does not match existing iteration node	The preprocess program changes the name of the current iteration. This is invalid. Correct the preprocess program code.
EventException017	Postprocess – Updated response iteration node does not match existing response iteration node	The postprocess program changes the name of the current iteration. This is invalid. Correct the postprocess program code.

Connection Exceptions

These exceptions are related to errors raised when using the Connection Pool Manager. These exceptions are included in the response QDocs.

Table 16.3 Connection Exceptions

Exception Code	Description	Notes
ConnectionException001	Unable to initialize Connection Pool	Cannot initialize a connection pool session. The maximum number of sessions has been reached, or the MFG/PRO databases are not up. There may also be a problem with the log-in script or a step omitted. Test this by following the log-in script from the Connection Pool setup parameters and ensuring all steps are correct. Another possible cause is that the startup script for launching the MFG/PRO session (invoked by the Connection Manager) may have been configured to run in normal mode instead of MFGWrapper mode. (See "Starting QGen" on page 114 for more on the MFGWrapper mode.) Review the startup script and update if incorrect, then restart the Connection Pool.
ConnectionException003	Failure to initialize Connection Pool Manager	The Connection Pool Manager cannot initialize. One responsibility of the Connection Pool Manager is to open XML configuration files. This error can occur if there is an invalid configuration file. To see a list of configuration files look at `config-files.xml`.
ConnectionException004	Unable to get an idle session	A client requested a new program to be run and an idle session cannot be retrieved from the connection pool. Verify that the connection pool has not exceeded the maximum number of connections. It is also possible that the Connection Pool Manager was initializing new sessions that were not yet available.

Exception Code	Description	Notes
ConnectionException005	Unable to get session because pool has been shut down	A user attempted to create a new session or make a request on an existing one after the Connection Pool Manager was shut down. In this case, all connections are invalid and the Connection Pool Manager must be restarted.
ConnectionException006	User has reached max sessions allowed	A user has requested to run another QDoc; however, this user has exceeded the maximum number of allowed connections.
ConnectionException007	Extracting a value from configuration file failed	Invalid value encountered during load of `connectionManagerConfig.xml`. Review all numeric values in the file.
ConnectionException008	Unable to locate the Connection Pool Manager configuration file	This file should be located in `webapps/qxtendserver/WEB-INF/config`. If it is not there, then a default version of the file can be copied and configured. This is located in `webapps/qxtendserver/WEB-INF/config/defaults`
ConnectionException009	Telnet connection not available	The telnet server for the specified receiver is unavailable.
ConnectionException010	Host is unavailable or incorrect	View the connection pool details for this receiver and validate the host is correct and available.
ConnectionException011	Telnet connection is refused	The connection to the host was refused. Manually launch a telnet connection from the QXtend machine to the host machine to isolate the problem.
ConnectionException012	MFG/PRO server is unreachable	The connection to the host is successful but the launch of an MFG/PRO session is not. Manually launch a telnet connection from the QXtend machine to the host machine and run the QXtend startup script to isolate the problem.

Queue Exceptions

If an exception is raised during QXtend startup, the UI paints the Queue link in red. This indicates an error that is logged to `queue.log` and `qdocInstall.log`. If an exception is raised during the processing of a QDoc through a queue, this exception appears in the `qdocinfo.log` and possibly the `qxtendserver.log`, as well as in the response QDoc.

Table 16.4
Queue Exceptions

Exception Code	Description	Notes
QueueException001	Unable to create queue object components	Class file names required to determine what Java classes to create are incorrect in `directoryloaderconfig.xml`.
QueueException002	Configuration file does not contain a queue node	The queue configuration file is missing a queue node. Update or create a new queue in the Queue Manager.
QueueException004	Error due to incorrect URL	The URL specified by the user in Update Queue in the Queue Manager is incorrect. The URL is the location of the QXtend Web services component.
QueueException005	Could not load Queue Manager config file	Verify the correct path and existence of: `directoryloaderconfig.xml`.
QueueException006	Could not create class	See QueueException001.
QueueException007	Could not locate `queueConfig.xml`	The queue configuration file has been deleted, moved, or renamed. Update or create a new queue in Queue Manager. Verify correct path and existence of: `queueConfig.xml`.
QueueException008	No queue type in queue configuration file	QXtend Inbound has one valid queue type—qdoc—in Queue Configuration. This exception is raised if the queue type is set incorrectly.
QueueException009	No queueClass element in configuration file	The settings for all queues are stored in an XML configuration file. This file must have been manually updated incorrectly. Using the Queue Manager, delete and recreate the queue to correct this.
QueueException010	No URL element in configuration file	The URL is set incorrectly in the Queue Configuration.
QueueException011	No frequency element in configuration file	The Frequency is set incorrectly in the Queue Configuration.

Exception Code	Description	Notes
QueueException012	Base directory does not contain any queues	No queues exist in the base directory. Create a new queue using the Queue Manager.
QueueException013	Envelope details missing from configuration file	SOAP envelope details required for QDoc updates are missing. This only impacts queues configured to add SOAP envelopes to the QDocs placed in the request directory. Update or create a queue.
QueueException014	No QDoc nodes in the default envelope	See QueueException013 for more information. Update the queue.
QueueException015	Queue configuration file missing	The queue configuration file, `queueConfig.xml`, has been deleted, moved, or renamed. Update or create a queue.
QueueException017	Specified queue missing	The queue directory has been removed or renamed. Create a new queue.
QueueException018	Queue Manager configuration file missing	The Queue Manager config file, `directoryloaderconfig.xml`, is missing. This file contains global configuration details.
QueueException019	QueueType node missing from configuration file	QueueType identifies the type of the requests on the queue. Update or create a queue.
QueueException020	No queueType or blank queueType specified in queue configuration file	The Queue Manager config file is incorrect. Open `directoryloaderconfig.xml`. Copy in the XML node shown under "Queue Type Node" following this table.
QueueException021	No queueTypes set up in queue config file	See QueueException020.
QueueException022	Queue Manager base directory missing	The full path to the top-level queue directory must be specified in `environmentmanager.xml`.

Queue Type Node

The following queue type node applies to QueueException020.

```
<queueTypes>
    <!-- Each queue Type supported by the directory loader must be
    specifed here. Each type then has a  processor class and a queue
    class associated with it-->
    <queueType>
        <!-- type - Identifies the type of a queue.
        processorClass - This is the class that will be used to
        process any requests on a queue of this type
        queueClass - This is the class that will be created to hold a
        queue of this type -->
        <type value="qdoc"/>
        <processorClass value=
        "com.qad.qxtend.utils.QdocFileProcessor"/>
        <queueClass value=
        "com.qad.qxtend.utils.QdocDirectoryQueue"/>
    </queueType>
</queueTypes>
```

SOAP Exceptions

These exceptions relate to the SOAP messages such as calling a SOAP service or extracting QDocs from the SOAP message. These exceptions appear in the response QDoc.

Table 16.5
SOAP Exceptions

Exception Code	**Description**	**Notes**
SOAPException001	Incorrect destination URL for SOAP request	The destination URL that points to the QXtend Inbound server is invalid or the URL is unreachable.
SOAPException003	SOAP manager configuration file missing	Verify the path and existence of `soapconfig.xml`.
SOAPException004	XML document create failed	Invalid XML in `soapconfig.xml`. Review and correct the file.
SOAPException005	Invalid data in SOAP configuration file	Invalid data in `soapconfig.xml`. Review and correct the file.
SOAPException006	Problem returning response	Writing QDoc response file to the SOAP response failed. Likely cause is that the HTTP connection has abnormally closed. View `queue.log`.

Exception Code	Description	Notes
SOAPException007	Request from document processor to pass to destination failed	Retrieving the QDoc response file from the SOAP request failed. Likely cause is that the HTTP connection has abnormally closed. View `queue.log`.
SOAPException008	No SOAP configuration file specified	When launching QXtend, if the Queue Manager is active, there is an entry in `environmentmanager.xml` for the SOAP caller manager. One of the nodes is the path to `soapconfig.xml`. Verify the path. View `qdocInstall.log`.
SOAPException009	Server is suspended	QXtend is suspended, possibly as a result of someone updating the configuration. Re-activate using Resume in QXtend.

Transaction Exceptions

Transaction exceptions relate to the QXtend transaction unit, which is made up of the request and response QDocs and manages the process of routing the message through either the UI API adapter or the code API adapter. These exceptions appear in the response QDoc.

Table 16.6 Transaction Exceptions

Exception Code	Description	Notes
TransactionException001	Error during processing. Full transaction roll back	QDoc processing failed. The entire transaction rolls back.
TransactionException002	Invalid routing ID	An invalid routing ID was used in the QDoc. The routing ID determines if the QDoc is processed using the code API or the UI API. Check the routing for each QDoc in the Configuration Manager under Schemas by MFG/PRO version.
TransactionException003	Unable to load routing document	The correct document is not in `routing.xml`, or is an invalid XML document.
TransactionException004	Error creating RoutingTask object	RoutingTask object cannot be instantiated. Review `routing.xml` for the QDoc for a particular route.

Exception Code	Description	Notes
TransactionException005	RoutingTask object is not a routing task	The RoutingTask object in `routing.xml` is not the correct type.
TransactionException006	Null field returned by the processor	MFG/PRO reports that the current field is null, or the session that started the program failed to initialize, or MFG/PRO aborted abnormally. To determine the cause, run an MFG/PRO CHUI session manually using the Connection Pool Manager startup script and follow the processing in the QDoc.
TransactionException007	Error during processing. Partial transaction roll back	QXtend was unable to complete the processing of the QDoc. View the response QDoc for the error that caused the problem. This rolls back the current transaction (main iteration level) in MFG/PRO.

Adapter Exceptions

This exception traps any undiagnosed exceptions during the processing of a QDoc using the UI API adapter. These exceptions appear in the response QDoc.

Table 16.7
Adapter Exceptions

Exception Code	Description	Notes
AdapterException003	Unknown exception servicing QDoc request	An undiagnosed exception was returned during QDoc processing. Review the response details for an indication of the problem. This exception should never occur. Contact QAD Support.

QDoc Exceptions

These exceptions report errors in the contents of request or response QDocs. This section also contains exception details when using the Queue Manager to process QDocs. These exceptions appear in the response QDoc.

Table 16.8
QDoc Exceptions

Exception Code	**Description**	**Notes**
QDocException001	Request QDoc missing SOAP envelope	Cannot read the QDoc SOAP envelope. Ensure the SOAP envelope is correct and validate the QDoc.
QDocException002	QDoc receiver is not recognized	Ensure the QDoc is set up for the correct receiver and the receiver is set up correctly.
QDocException003	Request does not contain a request body	QDoc must have a body. Review and correct the QDoc.
QDocException004	Failed to create a new XML document instance	Loading the QDoc schema file failed because the schema does not exist or the schema file does not contain valid XML. Check the schema version in the QDoc and validate the schema file.
QDocException005	QDoc is not supported	Ensure the QDoc is set up for the correct receiver and the receiver is set up correctly; add the appropriate schema to the receiver.
QDocException008	Error creating response QDoc	QXtend cannot create a QDoc response. Usually due to invalid XML data.
QDocException010	Failed to create a new QDoc response instance	See QDocException008.
QDocException011	Failed to parse the XML schema	See QDocException004.
QDocException012	Could not read schema file *fileName*	See QDocException004.
QDocException014	Failed to locate an initialized Connection Manager instance	The Connection Pool Manager is not initialized correctly. This should only occur on the first use of QXtend. Review Connection Manager, `connectionPools.log`, and `qdocInstall.log`.
QDocException015	Timed out waiting for idle AppServer connection	This applies to the code API adapter. Not enough application servers are available for the number of requests. Review the connections in the Connection Pool and update the number of sessions or timeouts.

Exception Code	Description	Notes
QDocException016	Failed to parse the QDoc schema	While processing a QDoc in the code API adapter or updating the response QDoc when `suppressResponseDetails` is set to false for the UI API adapter, the respective request and response schemas are loaded into memory and parsed. If an error occurs parsing these schemas, this exception is returned. Review the schemas and the exception detail.
QDocException017	Failed to map the QDoc request into a QdocResultSet	See QDocException016.
QDocException018	Failed to construct a new IQDocMapper instance	`qxtend.jar` does not contain the required classes. Contact your system administrator.
QDocException019	Processing of request on Progress AppServer connection failed	This applies to the code API. An unexpected exception occurred during the processing. See exception detail and logs.
QDocException020	QDoc processing failed with an unhandled exception	This applies to the code API. An unexpected exception occurred during the setup or processing of the QDoc. View the response QDoc and log files for further information.
QDocException021	Error on startup of the QXtend server	QXtend startup failed to launch all of the managers, such as Configuration, Connection Pool, Queue, and so on. View `qdocInstall.log`.
QDocException023	Failed to create data in the QDoc response	This applies to the code API. Creating the response failed. View the response QDoc and log files for further information.
QDocException024	Reading data from a result set returned from the AppServer failed	Failed to extract the response returned from Progress, or failed to add error data returned from the generic code API adapter to the response QDoc. View exception details.
QDocException025	Request prevented from being passed to the API	No response data has been returned from the code API adapter. View the QDoc response.

Exception Code	Description	Notes
QDocException026	Error during an API request	This exception returns Progress errors from processing QDocs through the code API adapter to the QDoc response.
QDocException027	Failed to locate specified file	A schema or events file was not located for a QDoc. Review the response QDoc and ensure the files are in the correct location.

QDocException028	QAD and custom namespaces are identical	Custom QDocs must have a unique, non-QAD namespace. View `implementationQdocs.xml` in the relevant receiver directory to check the namespace. The default namespace for QAD QDocs is:
		`<qadQdocs xmlns:qad= "http://www.qad.com/qdoc/eb">`
		For custom QDocs the default is:
		`<implementationQdocs xmlns:impl= "http://www.qad.com/customqdoc /eb"/>`

QDocException029	Could not extract header from SOAP envelope	The SOAP envelope header does not exist or is in the incorrect format. See “QDoc Message Envelope” on page 221 for sample SOAP envelopes.
QDocException030	Could not extract body from SOAP envelope	The body of the SOAP envelope does not exist or is in the incorrect format.
QDocException031	Could not set header in SOAP envelope	The SOAP header was not imported into the response QDoc.
QDocException032	Could not set body in SOAP envelope	The SOAP body was not imported into the response QDoc.
QDocException033	Request file cannot be found in request queue	This applies to the Queue Manager. A request QDoc was removed from the directory queue during processing.
QDocException034	Error processing request	This applies to the Queue Manager. This is a serious technical exception caused by an installation or environment problem, and should never occur. Review `queue.log` and `qxtendserver.log`. Contact your system administrator and QAD Support.

Exception Code	Description	Notes
QDocException036	Error getting request	This applies to the Queue Manager. The Queue Manager polls the request directory for a list of files, then loads the files for processing. If a request QDoc is deleted between these events, this exception is returned. View the `queue.log`.
QDocException037	No request directory found in path	This applies to the Queue Manager. The path to the request directory is invalid. Check `environmentmanager.xml`. Also check that a `requests` directory exists in the relevant queue folder.
QDocException039	Error writing response to file	This applies to the Queue Manager. QXtend was unable to write the response QDoc to the responses directory. This may be due to a permissions issue, or the directory was deleted or moved. Verify the response directory exists with correct permissions.
QDocException040	Request file is not in a working state	This applies to the Queue Manager. Moving a file that has not been processed to the responses directory failed. This occurs if duplicate file names are used.
QDocException041	Error writing contents of request file	This applies to the Queue Manager. QXtend is unable to write the request QDoc to the response directory. This is a permissions issue or the directory was deleted or moved. Verify the response directory exists with correct permissions. It is also possible that the response data is invalid. View `queue.log` and `qdocInfo.log`.
QDocException042	Error renaming file	This applies to the Queue Manager. QXtend is unable change the request QDoc to a processing state. This may be due to a permissions issue. Check permissions in the request directory. View `queue.log` and `qdocInfo.log`.

Exception Code	Description	Notes
QDocException043	Unreachable destination, invalid request, missing SOAP envelope	This applies to the Queue Manager. The Web service is down or the link is incorrect. View the request and response QDocs, `queue.log`, and `qdocInfo.log`.
QDocException044	Problem wrapping request in envelope	This applies to the Queue Manager. Adding a SOAP envelope to the QDoc through the Queue Manager failed. This may be due to a permissions issue. Verify correct permissions on the request QDoc.
QDocException045	Error occurred while adding the envelope	This applies to the Queue Manager. See QDocException044.
QDocException046	Error occurred when parsing the response	This applies to the Queue Manager. Writing the response QDoc to the response directory failed. A default QDoc response is created and written to the responses directory error detail. A permissions problem with the queue directories is also possible. Check `queue.log` and `qdocInfo.log` for more information.
QDocException047	Queue Manager is in error	This applies to the Queue Manager. QXtend cannot access the Queue Manager. View `queue.log`, `qdocInfo.log`, and `qdocInstall.log` for details.
QDocException048	Error launching Queue Manager	This applies to the Queue Manager. An exception was raised during the startup of the Queue Manager and is logged to `queue.log` and `qdocInstall.log`. The Queue UI link appears in red.
QDocException049	Could not find response file	This applies to the Queue Manager. Queue Manager is unable to write the response document to the response directory. Verify that the directory exists. View `queue.log` and `qdocInfo.log`.

Exception Code	Description	Notes
QDocException050	Could not create response document from QXtend output	This applies to the Queue Manager. Retrieving response details from QXtend failed. A default QDoc response is created and written to the response directory with error details. View `queue.log` and `qdocInfo.log`.
QDocException051	Could not find request file	See QDocException036.
QDocException052	Could not write XML to file	See QDocException039.
QDocException053	Could not create directory	QXtend Inbound is unable to create a directory during the configuration of a queue or receiver. This is possibly a permissions problem. Verify permissions on the corresponding directories.
QDocException057	Could not read request file	The request QDoc has incorrect format or contains invalid characters.
QDocException058	Could not write to request file	See QDocException041.
QDocException059	Import or include schema has invalid namespace	A node in the QDoc schema does not have an associated namespace.
QDocException060	Import or include schema has invalid schema location	An import schema in the QDoc is invalid. Ensure the QDoc notation is correct.
QDocException061	Invalid or missing requestor node	The useQDocRequestor entry in `qxtendConfig.xml` determines whether the requestor in the QDoc is used to process the QDoc. If this error is raised, useQDocRequestor is set to true and the QDoc SOAP envelope body does not contain a valid requestor node. Ensure that the requestor node contains both ID and password attributes.
QDocException062	Requestor type is required but undefined	User has a license other than an unlimited license and the QDoc SOAP envelope body does not contain a valid requestor node with a valid type attribute.

Configuration Exceptions

These exceptions relate to errors which may occur while configuring QXtend Inbound. These exceptions are displayed on the UI when configuring the system; more information is available in `qxtendserver.log` and `qdocInstall.log`.

Table 16.9 Configuration Exceptions

Exception Code	Description	Notes
ConfigurationException001	Receiver specified is not valid	While adding a new schema, the receiver assigned does not exist.
ConfigurationException002	Receiver directory not present	While deleting a receiver, the receiver directory does not exist.
ConfigurationException003	Reloading managers failed	QXtend launches managers that carry out specific functions, for example, Queue Manager to allow QDoc processing through a queue and Configuration Manager to configure QXtend Inbound. If QXtend Inbound is restarted using the Tomcat admin functionality or the restart option in QXtend, these managers are reloaded. If some configuration has changed prior to reloading which invalidates any of the managers, then the QXtend UI link displays red and this exception appears in `qdocInstall.log` with more information.
ConfigurationException004	Schema descriptor info incorrect	When modifying a schema, the schema details are retrieved from a configuration file. If the details are unavailable, this error is raised. This occurs if the configuration file is directly modified and that record deleted. Review the configuration for this schema and reinstall if necessary.
ConfigurationException005	Events directory not found	While adding or updating a schema, the events directory does not exist.
ConfigurationException008	Events file not found	When adding or modifying a schema, the events file must exist in the specified location. If not, this exception is raised.

Exception Code	Description	Notes
ConfigurationException009	Error occurred configuring the system	An exception occurred on QXtend startup related to the configuration of QXtend. The QXtend link displays red. The error is logged in `qdocInstall.log`.
ConfigurationException010	Error getting API configuration details	This error displays on the UI if an error occurs removing API support for a specified schema. This should only occur if the configuration files were manually updated. Contact the system administrator or QAD Support.
ConfigurationException011	Duplicate entries for an API	`QADQdocs.xml` or `implementationQdocs.xml` were incorrectly modified. View logs for more information and contact the system administrator or QAD Support.
ConfigurationException012	Referenced node missing in descriptor file	See ConfigurationException004.
ConfigurationException013	Error loading configuration details	An error occurred loading a configuration file. The file may have been manually moved or deleted, or manually updated with invalid data. View logs for more information. Also verify the path and file permissions.
ConfigurationException014	Schema files do not exist in specified directory	While adding or deleting a schema, the schema files cannot be located. Verify path and permissions.
ConfigurationException016	File already exists	While creating or modifying a schema, the descriptor file already exists. Verify path and permissions.
ConfigurationException017	Problem writing to new file	While adding a new schema, the schema files already exist.
ConfigurationException018	Duplicate nodes exist in descriptor file	You cannot create an entry in a descriptor file for a node that already exists.
ConfigurationException019	Schema directory not found	While adding a schema, the schema directory does not exist or is read-only. Verify path and permissions.

Exception Code	Description	Notes
ConfigurationException020	Deleting receiver directory failed	While deleting a receiver, the write to the backup directory failed.
ConfigurationException021	Receiver name cannot be null	While adding a receiver, the receiver name is blank or null. The null state may be raised if an error occurs recovering the value from the UI. Refresh and restart.
ConfigurationException022	Version not installed	While adding a receiver, the schema directory does not exist, could not be created, or the receiver version is blank.
ConfigurationException024	Receiver already supported	You cannot add a new receiver that already exists for that version. Specify a different receiver or edit existing receiver.
ConfigurationException025	Receiver details not found	While deleting a receiver, cannot find the receiver node in `receivers.xml`.
ConfigurationException026	Parsing descriptorInfo failed	Building a hashmap of nodes from a descriptor file failed. Verify that the file exists.
ConfigurationException027	Updating descriptorInfo failed	This error occurs while creating a new receiver structure, path directory, or subdirectory for requests. It can also occur while updating global or receiver descriptor files with a new node.

Process Exceptions

These exceptions relate to errors which may occur during the processing of a QDoc. These exceptions appear in the response QDoc and/or in log files.

Table 16.10
Process Exceptions

Exception Code	Description	Notes
ProcessException001	Session failed to initialize	QDoc processing program failed to launch. Review the Connection Pool Manager settings for the pool and test the initialization by manually following the pool log-in script.
ProcessException002	Failure to send message to begin submitting data	Data submission to the server failed. An acknowledgement may fail due to the trigger failing to fire. To find the root cause, follow the process through a character session and look for any unusual user interface functions such as alert boxes, selection lists, or browses.
ProcessException003	Failure to send action or data message	See ProcessException002.
ProcessException005	Failure to send spacebar event	QXtend is unable to respond to a pause event that occurs in MFG/PRO. Enable QXtend logging to the warn level and follow the process through a character session, looking for any unusual interface functions around that field. See “Log Report Levels” on page 53.
ProcessException007	Java encoding specification sent from Progress failed	The MFG/PRO session request for user encoding from QXtend returned an invalid encoding string. Check the user encoding. This exception may also occur if the session failed to initialize correctly. Check the initialization procedure described in ProcessException001.
ProcessException009	Timeout occurs waiting for response from MFG/PRO QdocReceiver Configuration Manager	Communication occurs constantly between QXtend and MFG/PRO, from session start to the processing of the QDoc. QXtend Inbound defines a timeout for a response to take place from MFG/PRO. If the expected response is not received within this timeout period, this exception is returned. May occur if a record is locked or if the terminal was exited abnormally.
ProcessException010	Error while attempting to reset a Progress AppServer connection	Closing the connection to the application server raises an error. Check the details on the exception for more information.

Failure Exceptions

These are low-level exceptions, which, in most cases, are wrapped within another exception and are only seen at the lower level of a stack trace. These exceptions appear in the response QDoc and/or in log files.

Table 16.11
Failure Exceptions

Exception Code	**Description**	**Notes**
FailureException003	Writing document to file failed	While updating configuration settings using the Connection Pool Manager, the new settings cannot be written to a file.
FailureException005	Loading environmentmanager manager classes failed	QXtend startup failed to launch all of the managers, such as Configuration, Connection Pool, Queue, and so on. View `qdocInstall.log`.
FailureException006	New XML document instance create failed	This error occurs if QXtend attempts to read in an invalid XML document.
FailureException007	Unknown error occurred instantiating manager class	An error occurs starting up QXtend. This may be caused by invalid configuration files. View `qdocInstall.log`.
FailureException008	Unable to find or load configuration file for manager class	A configuration file was not found during QXtend startup. View `qdocInstall.log`.
FailureException009	Unable to find or load configuration file	A configuration file listed in `config-files.xml` was not found during QXtend startup. View `qdocInstall.log`.

Glossary

4GL. An abbreviation for fourth generation programming language, such as the Progress language. Direct APIs to the Progress code are more efficient than calls to the user interface.

API. application program interface. A set of routines, protocols, and tools that connect applications, usually for the purpose of sharing data.

B2B. Business-to-business; the exchange of products, services, or information between businesses rather than between businesses and consumers.

CRUD. Create-read-update-delete. Used to describe a software application that can perform those actions on database records.

ebXML. See *Electronic Business XML (ebXML)*.

Electronic Business XML (ebXML). A joint project by the United Nations body for Trade Facilitation and Electronic Business Information Standards (UN/CEFACT) and the Organization for the Advancement of Structured Information Standards (OASIS) to use XML to standardize the secure exchange of business data.

Emulation. See *Terminal Emulation*.

Encryption. Conversion of data into a form that cannot be easily intercepted by unauthorized people.

Events. A file generated during QDoc creation by QGen that contains field navigation information. Controls the processing of a QDoc request through the MFG/PRO user interface by indicating iteration levels and non-standard navigation. Each supported calling procedure has an associated events file.

Extensible Markup Language (XML). A specification designed especially for Web documents that allows the definition, transmission, validation, and interpretation of data between applications and organizations.

Extensible Style Language (XSL). A language for formatting an XML document; for example, showing how the data described in the XML document should be presented in a Web page.

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

HTTP (Hypertext Transfer Protocol). The set of rules for exchanging text, graphic images, sound, video, and other multimedia files on the World Wide Web.

Iteration. A repeatable data entry cycle in MFG/PRO, such as sales order lines, or the entire sales order. Defined in QGen as the cycle from the first field back to the first field.

Java. An object-oriented programming language created by Sun Microsystems. Java is a device-independent language. Programs compiled in Java can be run on any computer. Java programs can be run as free-standing applications or as applets placed on a Web page.

Java 2 Platform, Enterprise Edition (J2EE). A recent release of Java designed to support the requirements of large-scale computing systems. Features include Java servlets and Java Server Pages (JSPs), which facilitate dynamic Web-enabled data access and manipulation.

Java Development Kit (JDK). A software development environment from Sun Microsystems for writing applets and applications in the Java programming language.

Java Message Service (JMS). An API from Sun Microsystems that supports formal communication—or messaging—between computers in a network.

Java Plug-in. Software provided by Sun Microsystems that replaces the default virtual machine associated with a Web browser. Using the Java plug-in allows developers to deploy Java applets that depend on the latest features of the Java platform and be assured that their applets will run reliably and consistently in both Microsoft Internet Explorer and Netscape Navigator.

Java Runtime Environment (JRE). A subset of the Java Development Kit for end users and developers who want to redistribute the Java runtime environment. The Java runtime environment consists of the Java virtual machine (JVM), the Java core classes, and supporting files.

Java Server Page (JSP). A technology for controlling the content or appearance of Web pages through the use of servlets, small programs that are specified in the Web page and run on the Web server to modify the page before it is sent to the user who requested it.

Java Virtual Machine (JVM). The part of the Java runtime environment responsible for interpreting bytecode.

JDK. See *Java Development Kit (JDK)*.

JRE. See *Java Runtime Environment (JRE)*.

JSP. See *Java Server Page (JSP)*.

JVM. See *Java Virtual Machine (JVM)*.

Namespace. In XML, a unique identifier for a collection of element type and attribute names. In an XML document, any element type or attribute name can have a two-part name consisting of the name of its namespace and then its local—or functional—name.

PROPATH. An environment variable containing the list of directories Progress searches when looking for a program to execute.

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

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

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

Queue Manager. An optional interface to manage QDoc requests and responses through a specified directory structure. The Queue Manager can also add a SOAP envelope to a QDoc to support QXtend requirements.

Receiver. A receiver identifies MFG/PRO instances that process inbound QDoc requests.

Remote Procedure Call (RPC). A protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details.

Script. A program or sequence of instructions that is interpreted or carried out by another program.

Secure Sockets Layer (SSL). A program layer for managing the security of message transmissions in a network. The program layer exists between an application (such as a Web browser or HTTP) and the Internet's TCP/IP layers. Sockets refers to the sockets method of passing data back and forth between a client and a server program in a network or between program layers in the same computer.

Servlet. Programs similar to Java applets, that run on the server rather than the client and are used to run interactive Web applications.

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

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

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

SSL. See *Secure Sockets Layer (SSL)*.

TCP/IP. See *Transmission Control Protocol/Internet Protocol (TCP/IP)*.

Telnet. A user command and underlying TCP/IP protocol that lets you access applications and data on remote, or *host*, computers.

Terminal Emulation. Use of a personal computer to interact with a computer with a different operating system. The terminal emulation program runs as a separate task with its own window. The application interface presented in this window is character-based or text-only.

Tomcat. The servlet container used in the official reference implementation for the Java Servlet and Java Server Pages (JSP) technologies. Tomcat is developed in an open and participatory environment and released under the Apache Software Foundation license.

Transmission Control Protocol/Internet Protocol (TCP/IP). The basic communication language or protocol of the Internet. It can also be used as a communications protocol for intranets and extranets.

UI. See *User Interface (UI)*.

Uniform Resource Identifier (URI). A method of identifying or reserving a point of content on the internet, such as a page of text, a graphic image file, or a program. A URI typically describes:

- The mechanism used to access the resource
- The specific computer that the resource is housed in
- The specific name of the resource (a file name) on the computer

The most common form of URI is a uniform resource locator (URL).

Uniform Resource Locator (URL). A text string that indicates the location of an intranet or Internet resource.

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

User Interface (UI). The portion of an application that is visible to the user and the mechanism by which the end user interacts with the application, enters information into the application, and sees the results of the interaction.

UUID. See *Universal Unique Identifier (UUID)*.

W3C. See *World Wide Web Consortium (W3C)*.

WAR. See *Web Archive File (WAR)*.

Web Archive File (WAR). A compressed file containing a Web application and its related files. Assists in easily deploying an entire application.

Web Services. Vendor-neutral, XML-based, remote procedure call (RPC) protocol that allows any system to run programs in other, dissimilar systems.

Web Services Description Language (WSDL). An XML-based language used to describe the services a business offers and to provide a way for individuals and other businesses to access those services electronically.

World Wide Web Consortium (W3C). An international industry consortium that seeks to promote standards for the evolution of the Web and interoperability among Internet products by producing specifications and reference software.

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

XML Schema Definition (XSD). An abstract representation of the elements in an XML document that can be used to verify that each item of content adheres to the associated element's description. The XSD standard follows the W3C recommendation.

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

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

Index