QAD Enterprise Applications
Standard & Enterprise Edition

Training Guide

QXtend Fundamentals

70-3228A
QAD QXtend 1.7.2
QAD 2011.1 Standard & Enterprise Edition
Lab: Enterprise Edition 2011 - QXtend Training r01
September 2011

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

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

QAD and MFG/PRO are registered trademarks of QAD Inc. The QAD logo is a trademark of QAD Inc.

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

Copyright © 2011 by QAD Inc.

QXtendFundamentals_TG_v2011.1SE&EE.pdf/hes/mdf

QAD Inc.
100 Innovation Place
Santa Barbara, California 93108
Phone (805) 566-6000
http://www.qad.com

Contents

What's New? xi

About This Course 1

Course Description 2
Course Objectives 2
Audience 2
Prerequisites 2
Course Credit and Scheduling 2
Virtual Environment Information 2
QAD Web Resources 3

Chapter 1 QAD QXtend Overview 5

Overview 6
Enterprise Application Integration (EAI) 7
Interoperability 8
Service-oriented Architecture (SOA) 9
Enterprise Service Bus 10
QAD Enterprise Applications 11
Why QAD QXtend? 13
Data Synchronization 15
Exercise: Interoperability Overview 16
QAD QXtend Inbound 18
QDocs 23
Web Service 24
Receivers 25
Connection Pools 26
Adapters 28
User Interface Adapter 30
Service Interface (SI) Adapter 32
Fin API Adapter 33
APIs 34
QAD QXtend Outbound 37
Source Applications 42
Business Objects 43
Profiles 45

Subscribers 46
Data Flow 48
Exercise: QXO Concepts 49

Chapter 2 QXI Configuration 51

Overview 52
Receivers 53
Receiver Management 55
Add Receiver 56
Supported APIs 57
Receiver Report 58
Modifying a Receiver 59
Modifying a Receiver - Supported APIs 61
Deleting a Receiver 62
Exercise: Receivers 63
Connection Pools 64
Connection Pool Manager 66
Adding a Connection Pool 67
Adding UI API Connection Pool 68
Adding SI API Connection Pool 70
Adding an Fin API Connection Pool 72
Managing Connection Pools 73
Viewing a Connection Pool 75
Deleting a Connection Pool 76
Exercise: Connection Pools 77
Validating Your QXI Configuration 79
Testing QXI Using SOAP UI 82
Create New SOAP UI Project 87
Edit QDoc Request 88
Edit QDoc Request (Update Header) 89
Edit QDoc Request (Session Context) 91
Edit QDoc Request (Request Data) 93
Process Request 95
QXI APIs (Schemas) 97
API Types 98
Deleting a Custom API 102
Exercise: Schemas 103
Lab: QXI Configuration 105
1. Access and Start the Training Environment 105
2. Create the Receiver 106
3. Create a Connection Pool 107
4. Adding APIs to a Receiver 108
5. Processing a Request using the Test Harness 109

6. Load New API 118
7. Verify the Customer Item API has been added 120
8. Test the New Customer Item Maintenance API 121

Chapter 3 QXO Configuration 125

Introduction 126
Source Applications 127
Source Application Types 128
Creating a Source Application 130
Managing Database Connections 132
Managing Event Types 134
Exercise: Source Applications 139
Services 140
Event Service 142
Event Service Configuration 144
Threshold Alerts 147
Source Applications 148
Message Publisher 150
Configuring the Message Publisher 152
Business Objects 155
Subscribers 156
Configuring Subscribers 158
File Directory Service 159
QXtend Web Service 162
Web Service 164
Profiles 165
Source Applications 166
Message Sender 169
Configuring Message Senders 171
Subscribers 174
Viewing and Controlling Services 175
Monitoring QXO Processing 179
Event Viewer 180
Raw Messages Viewer 181
Subscriber Messages Viewer 182
Subscriber Messages - Request 184
Subscriber Messages - Response 185
Lab: QXO Configuration 186
1. License Manager Configuration 186
2. Source Application Types 187
3. Source Application 187
4. Load Default Business Objects and Profiles 189
5. Review Business Objects and Profiles 190

6. Create QXO Services 191
7. Configure Generalized Code Data Synchronization 194
8. QXI Receiver Configuration 199
9. Start the QXO Services 199
10. Test the Data Synchronization Configuration 200

Chapter 4 Licensing 203

Overview 204
License Manager 210
Licensing Configuration 212
Agent and Receiver Status 217
Licensing Reports 218
Exercise: QXtend Licensing 219
Lab: QAD QXtend Licensing 221
1. License Exceptions 221
2. Standard License Type 222
3. QXO Licensing 224
4. Monitoring License Usage 226
5. QAD QXtend Enterprise License 227

Chapter 5 QGen. 231

QGen Overview 232
Using QGen 233
Mapping 235
Generating API Files 244
Mapping Comments 250
QGen Extras 252
Exercise: QGen 253
Lab: QGen 255
1. Mapping Simple Screens 255
2. Mapping Complex Screens: Parent-Child Iterations 263
3. Mapping Complex Screens: Comments 267

Chapter 6 Business Objects and Profiles. 275

Business Objects 276
Business Objects Example 277
Using Business Objects 278
Creating a Business Object 279
Adding Fields 287
Options 289
Predefined Business Objects 291
XML Import/Export 292

Exercise: Business Objects 293
Profiles 295
Creating Profiles 298
QDoc Operations 305
Delta QDocs 308
XML Schemas 309
Exercise: Profiles 310
Lab: Business Objects and Profiles 312
1. Generalized Code Data Synchronization 312
2. Sales Order File Generation 318
3. Synchronize Item Master Data 323

Chapter 7 QXI Queue Manager 327

Overview 328
Initializing the Queue Manager 330
Adding a Queue 332
Modifying a Queue 336
Queue Management 337
Queue Monitoring 338
Deleting Messages 340
Viewing Requests/Responses 341
Viewing Request Details 342
Viewing Responses 343
Viewing Response Details 344
Editing/Resubmitting Requests 345
Exercise: Queue Manager 346
Lab: Queue Manager 348
1. Enabling Queue Manager 348
2. Configuring Queues 350
3. Process Transactions Using the Queue 352

Chapter 8 Service Interface Layer 357

Overview 358
Service Interface Adapter 362
Configuring the SIAPI 363
Enterprise Financials Inbound Integration 364
Configuring SIAPI in Enterprise Financials 365
Loading a Schema into Enterprise Financials 366
Enterprise Financials - Request Context 368
Examples 369
Modify and Delete Examples 371
Exercise: Service Interface Layer 373

Lab: Service Interface Layer 375
1. Fin API 375
2. SI API 381

Chapter 9 Direct Data Publish 385

Overview 386
Configuring DDP in QAD QXtend 388
Configuring the DDP Source Application Type. 390
Configuring the DDP Source Application 391
Configuring the DDP AppServer 392
Configuring DDP Business Objects 393
Enterprise Financials Outbound Integration 395
Direct Data Publish 396
Enterprise Financials - DDP Configuration 397
Configuring Enterprise Financials for DDP 399
Configuring the Application ID 400
Configuring the Event Daemon 401
Configuring the Event Destination 402
Configuring the Event Types 403
Publishing the Data 404
Checking the Event Daemon 405
Starting the Daemon 406
Generating an Application Event 407
Checking the Event Queue 408
Checking the QXO Subscriber 409
Exercise: Direct Data Publish 410
Lab: Direct Data Publish 412
1. Configure QXO for Financials DDP 412
2. Configure QAD Enterprise Applications 415
3. Publish Data from QAD EE to QXO 418

Chapter 10 Query Service 421

Overview 422
Calling Options 424
Web Service Invocation 425
QXI Configuration 426
Using Web Service 430
Exercise: Query Service 435
Lab: Query Service 436
1. QXO Configuration 436
2. Query Service – Call from QXI 438

Chapter 11 Diagnostics and Troubleshooting 445

QAD QXtend Inbound Logging 446
QAD QXtend Inbound Log Files 448
QDoc Requests Log 449
QDoc Response Log 450
QDoc Processing Log 451
QAD QXtend Server Install Log 453
Connection Pool Manager Log 454
QAD QXtend Server Processing Log 455
QAD QXtend Server Processing Log 456
Queue Manager Log 457
Debugging UI API QDocs 458
View Telnet Connection Pool 459
Telnet Screen - Current State 460
Viewing Multiple Telnet Windows 461
Extra Logging Parameters 462
QAD QXtend Inbound Exception Codes 463
QAD QXtend Outbound Logging 464
Background Sessions - Log Files 465
Identify Session Log File 466
Session Log Files: Log Level 467
Log Files - Sample for Event Service 468
Log Files - Sample for Message Publisher 469
Log Files - Sample for Sender 470
4GL Trace: start-sess.sh 471
QXO Viewers Tab 472
Exercise: QAD QXtend Diagnostics and Troubleshooting 473
Lab: Troubleshooting 475
1. QAD QXtend Inbound Logging 475
2. QAD QXtend Outbound Logging 477

What's New?

The following table summarizes significant differences between this document and the last published version.

Date/Version	Description	Reference
September 2011.1 EE	Rebranded for QAD 2011.1 EE	---

About This Course

Course Description

This course covers the basic concepts, setup, and use of the QAD QXtend interoperability platform. The Training Guide — QXtend course includes:

- A description of the terms and concepts that are specific to the product
- An overview of the QAD QXtend interoperability platform, whose primary components are QAD QXtend Inbound (QXI) and QAD QXtend Outbound (QXO)
- Exercises for configuring QXI and QXO, administering the QGen utility, using the logging functionality, and setting up tools such as Query Service

Course Objectives

By the end of this class, students will be able to:

- Describe the basic concepts of the QAD QXtend interoperability platform
- Configure the QAD QXtend License Manager and run licensing reports
- Use QGen to map QAD Enterprise Application screens
- Understand receivers, agents, connection pools, direct data publishing, QGen, Test Harness, and the Queue Manager
- Understand source applications, services, business objects, subscribers, profiles, and validate a QAD Outbound configuration
- Understand how to use the QAD QXtend logging functionality
- Set up and configure the Query Service

Audience

This course is designed for:

- Implementation consultants and teams
- Sales and account managers

Prerequisites

A knowledge of ERP systems or business applications in general, as well as interoperability-related models and technologies, is recommended.

Course Credit and Scheduling

This course is worth 30 course credit hours and usually requires five days for instructor-led training.

Virtual Environment Information

The hands-on exercises in this book should be used with the “Enterprise Edition 2011 - QXtend Training r01” environment.

QAD Web Resources

The QAD website provides product and company overviews. The Print Solution option on the opening page provides a means of compiling desired content into a document specialized to your industry, business implementation, and needs.

http://www.qad.com/

From the QAD main site, you can access the QAD Learning or Support sites.

Chapter 1

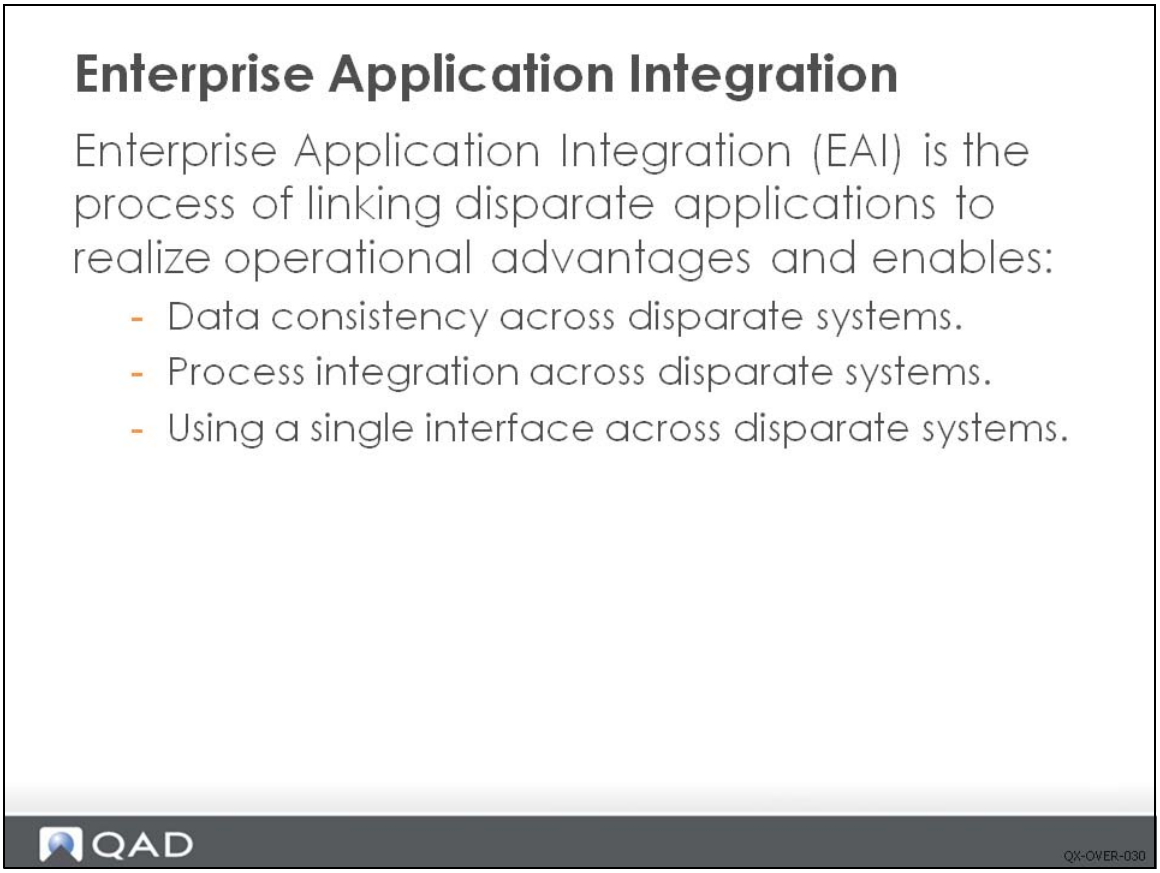
QAD QXtend Overview

Overview



This section discusses a number of concepts and their specific use in the context of the QAD QXtend interoperability platform. This section concludes with an exercise to test your understanding of the concepts.

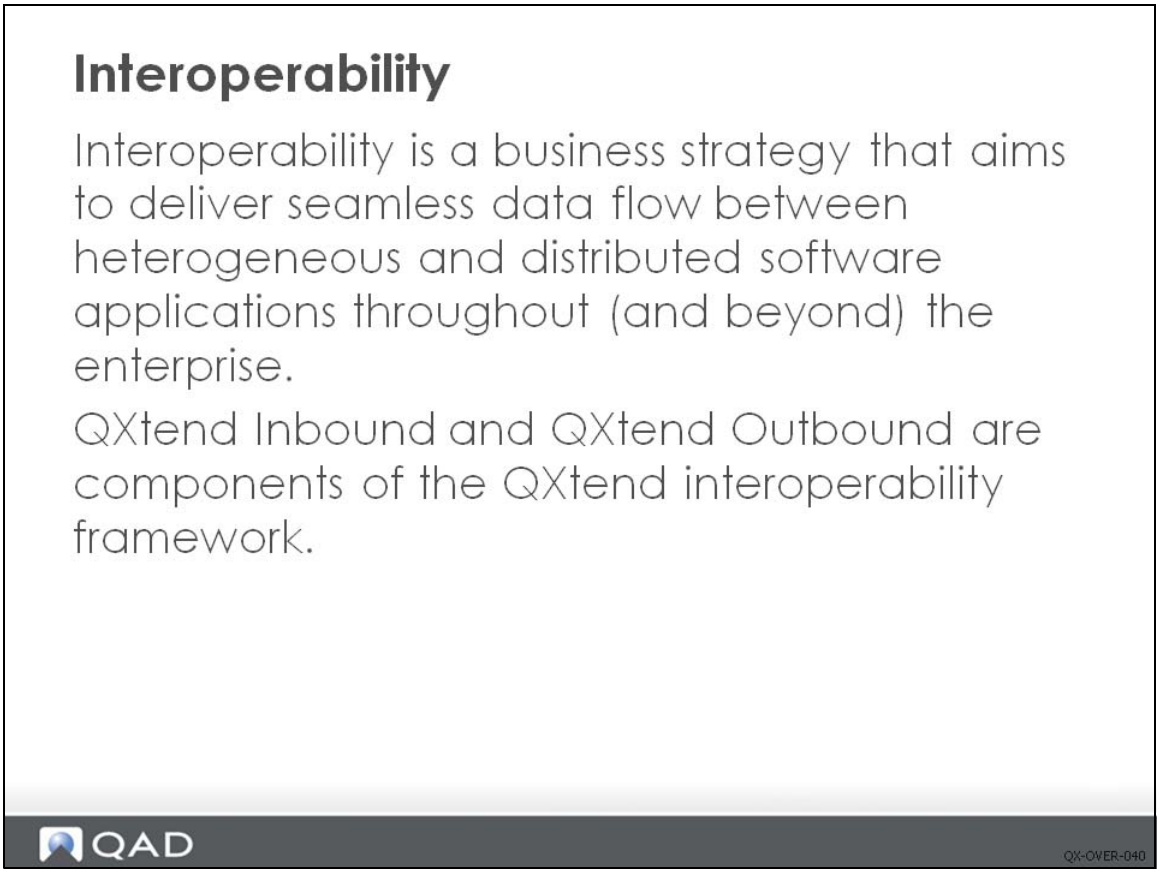
Enterprise Application Integration (EAI)



Supply chain management applications, customer relationship management (CRM) applications, and other types of applications typically cannot share information with one another. Such applications are sometimes consequently referred to as *islands of automation*. This inability to communicate leads to inefficiencies, such as the storage of identical data in multiple locations.

Enterprise application integration (EAI) is the process of linking applications within a single organization in order to avoid these inefficiencies, and to simplify and automate business processes as far as possible.

Interoperability

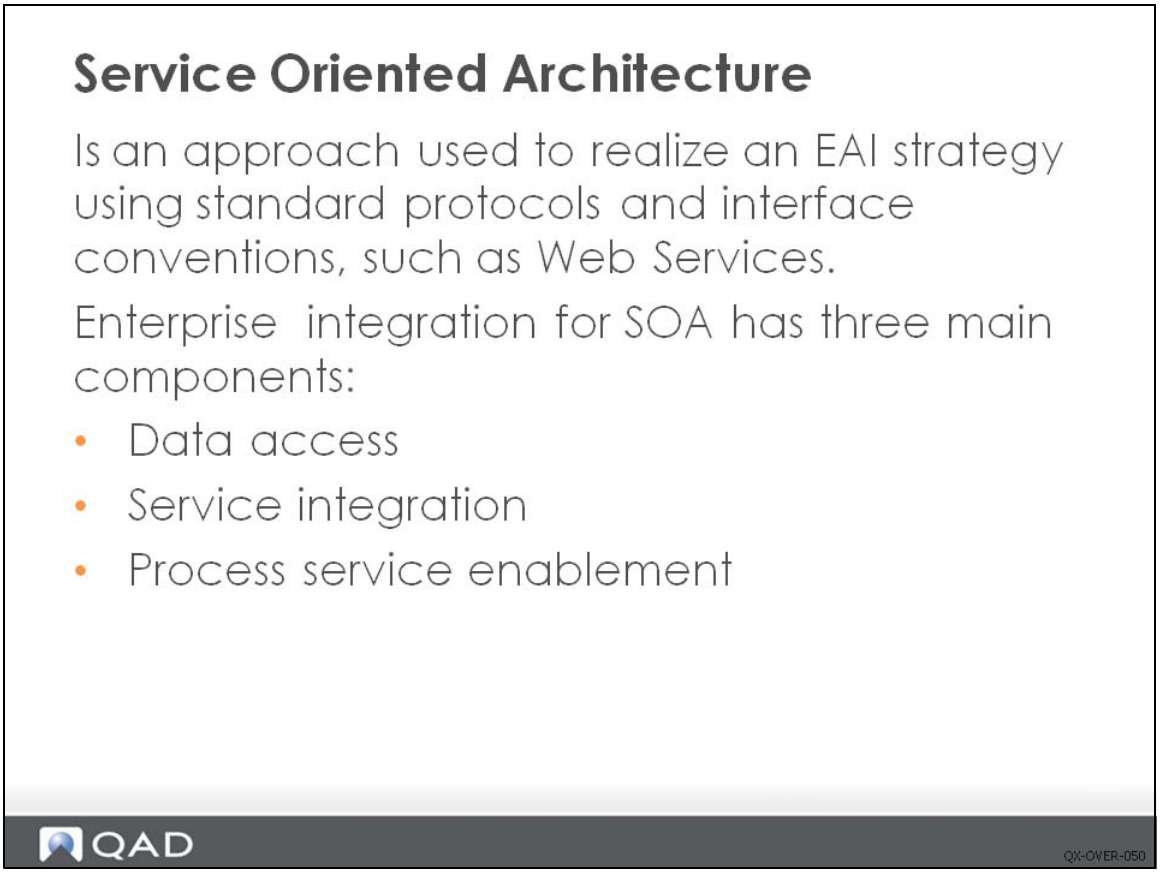


Interoperability is a business strategy that aims to deliver seamless data flow between heterogeneous and distributed software applications throughout (and beyond) the enterprise.

QXI and QXO are components of the QXtend interoperability framework, which 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 QAD Enterprise Applications business functionality.

Service-oriented Architecture (SOA)



Service-oriented architecture (SOA) describes an IT infrastructure that allows different applications to exchange data with one another when engaging in business processes.

SOA couples services with operating systems and other technologies that underlie applications. SOA separates functions into distinct units, or *services*, which can be accessed over a network so that the services can be combined and reused. Services communicate with other services by exchanging data.

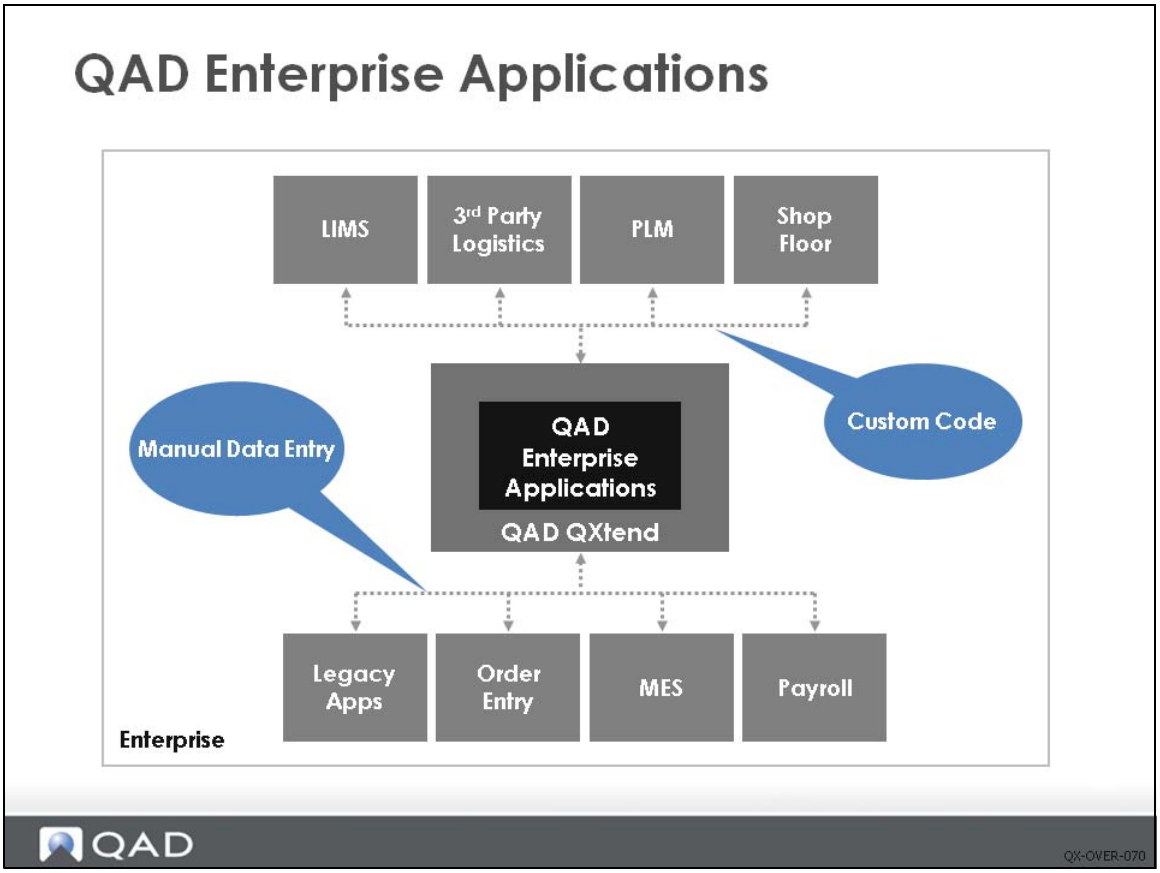
Enterprise Service Bus



The enterprise service bus (ESB) is a standards-based message and events management and orchestration framework that:

- Allows separation of integration from applications and business logic.
- Orchestrates services management.
- Orchestrates message routing, delivery, and transformation.
- Enables message tracking and integration monitoring.

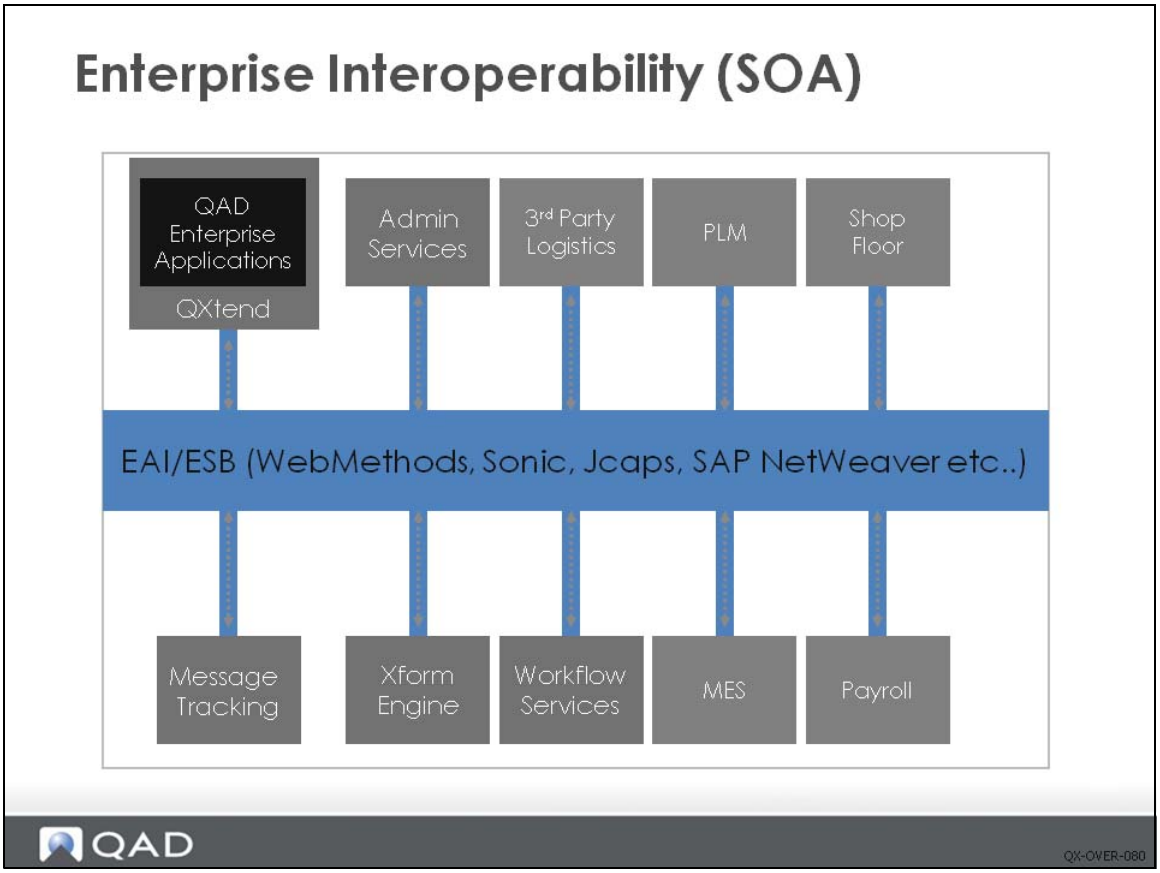
QAD Enterprise Applications



Interoperability enables data from disparate systems within a business enterprise to be shared with other systems with the enterprise without resorting to manual data entry or implementing custom code.

Manual data entry is a slow and time-consuming way to share data with other systems, often introducing errors that require correction. Custom code is expensive to create, quality assure, and maintain. Implementing QAD QXtend can remove these inefficiencies.

Enterprise Interoperability (SOA)

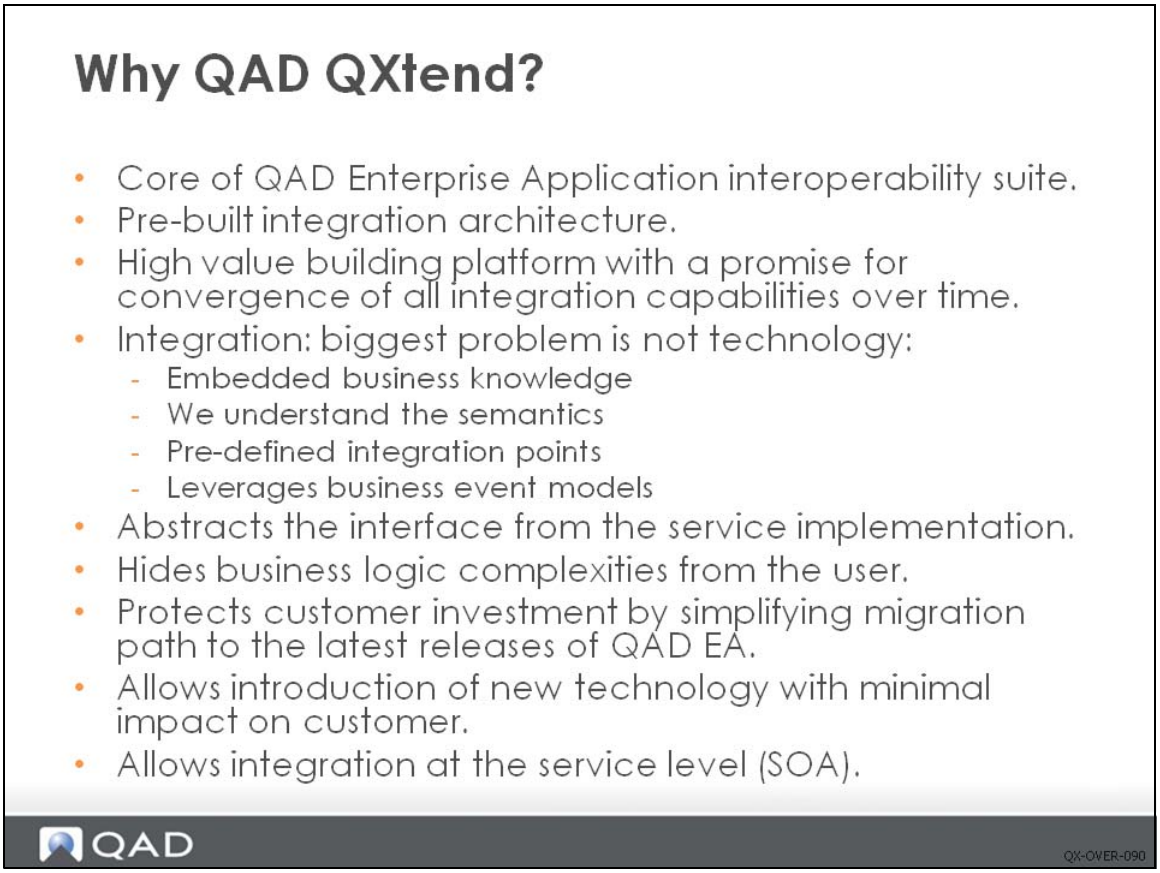


Implementing an EAI strategy that leverages the services of an ESB greatly simplifies enterprise interoperability. All applications in the enterprise use a common approach to achieve the required business solution. Applications only control the communication to and from the ESB, and all other orchestration and processing is controlled by execution rules in the EAI/ESB infrastructure.

The central EAI/ESB approach enables the construction of common components that perform services that are essential in any enterprise integration. Examples of such components are:

- Message tracking: This component tracks the flow of messages through a business process that is executed against many disparate applications. Tracking the messages at the EAI/ESB level provides visibility into the entire business process being executed and enables monitoring and management of those processes. The great advantage is that these services can be provided as a common component and reused by all applications that use the EAI/ESB.
- Xform (transformation) engine: This component transforms messages from one format to another. Having a transformation engine is a common requirement of any integration. By providing this as a service on the EAI/ESB layer, all applications can leverage it rather than each application controlling the mapping process.

Why QAD QXtend?

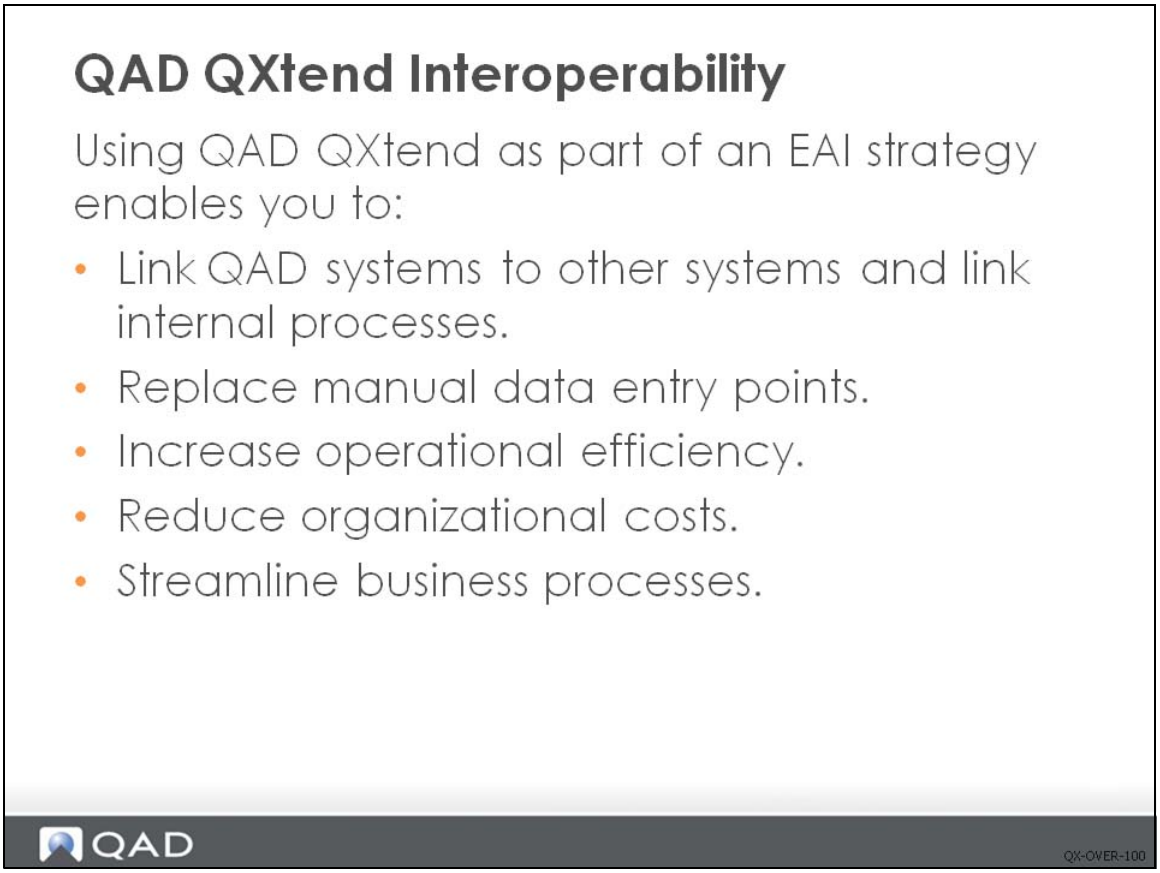


QAD QXtend is the core of QAD Enterprise Application interoperability suite. It is a prebuilt integration architecture with predefined integration points, and which leverages business event models.

In addition using QAD QXtend can provide the following advantages:

- Abstracts the interface from the service implementation helping to hide the business logic complexities from the user. QAD QXtend can change the way the service is implemented (rewrite the backend) if the user is concerned with how the backend has been implemented.
- Protects customer investment by providing a simpler migration path to the latest releases of QAD EA. This is achieved by removing custom code and relying on standard interfaces.
- Allows the introduction of new technology such as the service interface API with minimal impact on the customer.
- Allows customer to integrate at the service level (SOA).
- QAD QXtend represents the future development effort and long-term direction for QAD.

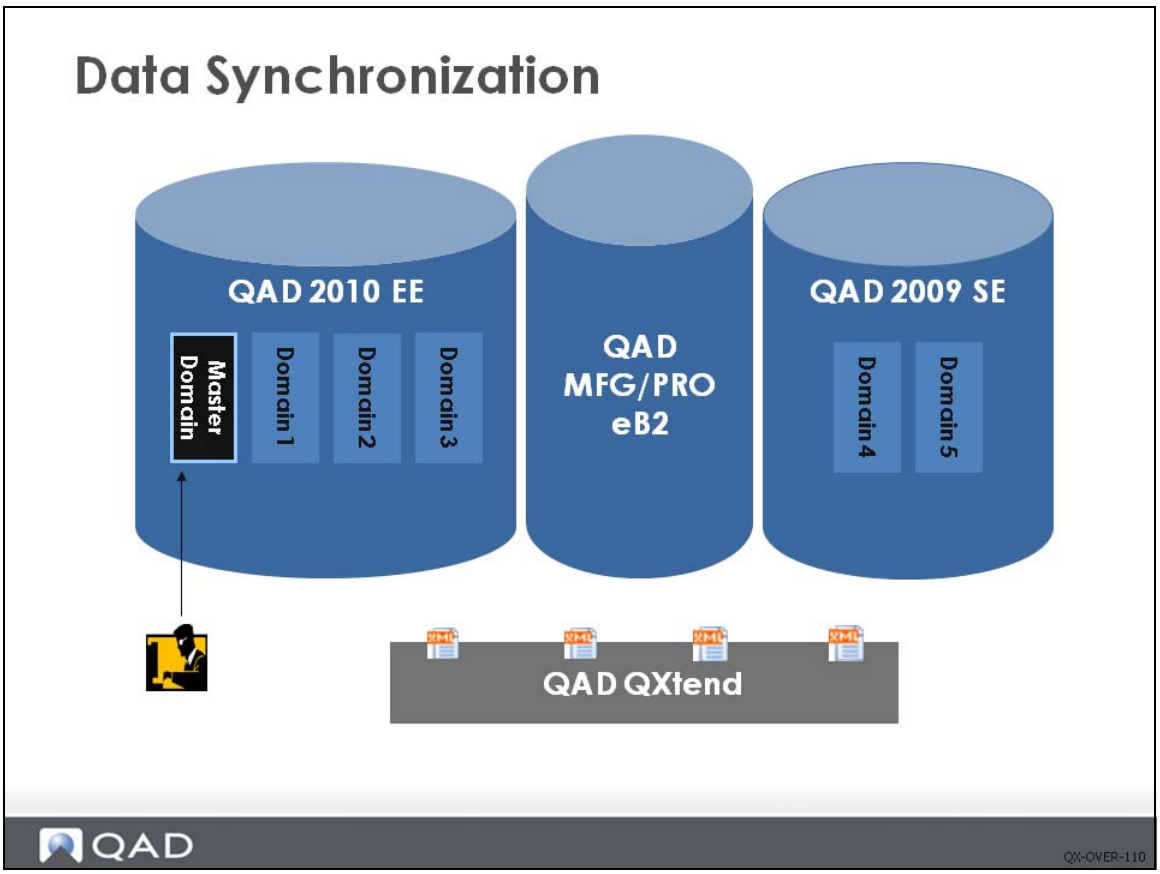
QAD QXtend Interoperability



Using QAD QXtend as part of an EAI strategy enables you to:

- Link QAD systems to other systems (legacy or vendor systems) as well as link internal processes.
- Replace manual data entry points that can lead to data integrity issues.
- Increase operational efficiency.
- Reduce organizational costs.
- Streamline business processes.

Data Synchronization



Data synchronization is the on-going process of establishing consistency among data on remote sources on disparate applications. QXtend can handle data synchronization between different version of QAD EA; for example, from a QAD EE instance to an instance of eB2, or vice versa, as shown above.

The diagram above illustrates how data synchronization operates:

1. Changes are made to data in the master domain.
2. Changes are imported as an XML document into QAD QXtend.
3. The XML documents describing the changes are propagated to domains in external applications, such as other instances of QAD EA, regardless of the version.

Exercise: Interoperability Overview



The following list shows a number of key terms and concepts used in interoperability and the QAD interoperability platform. In each statement below, fill in the correct term from the list.

data consistency	information
interoperability	EDI eCommerce
services	Data synchronization
services management	seamless data flow
integration	islands of automation

1 Enterprise Application Integration is the process of linking disparate applications to achieve _____ across disparate systems.

2 Another objective of EAI is to eliminate _____, also known as _____ silos.

3 Interoperability is a business strategy that aims to deliver _____ between heterogeneous and distributed software applications throughout (and beyond) the enterprise.

4 Service Oriented Architecture separates functions into distinct units, or _____, which can be accessed over a network so that the services can be combined and reused.

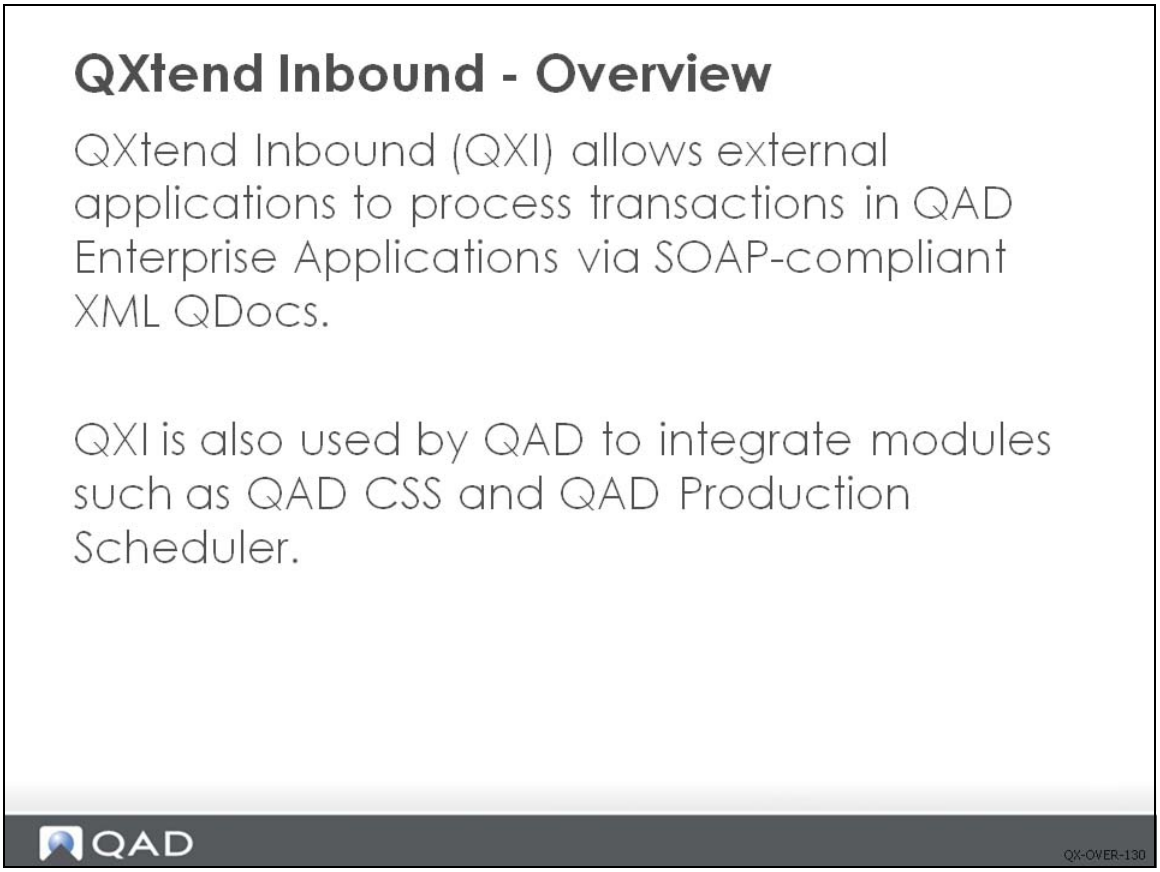
5 In addition to orchestrating message routing, delivery, and transformation, the enterprise service bus also orchestrates _____.

6 QAD QXtend is the core of the QAD Enterprise Applications _____ suite. It is a prebuilt integration architecture with predefined _____ points, and which leverages business event models.

7 Interoperability software like QAD QXtend and _____ deliver seamless data flow between heterogenous and distributed software applications throughout (and beyond) the enterprise.

8 _____is the on-going process of establishing consistency among data on remote sources on disparate applications.

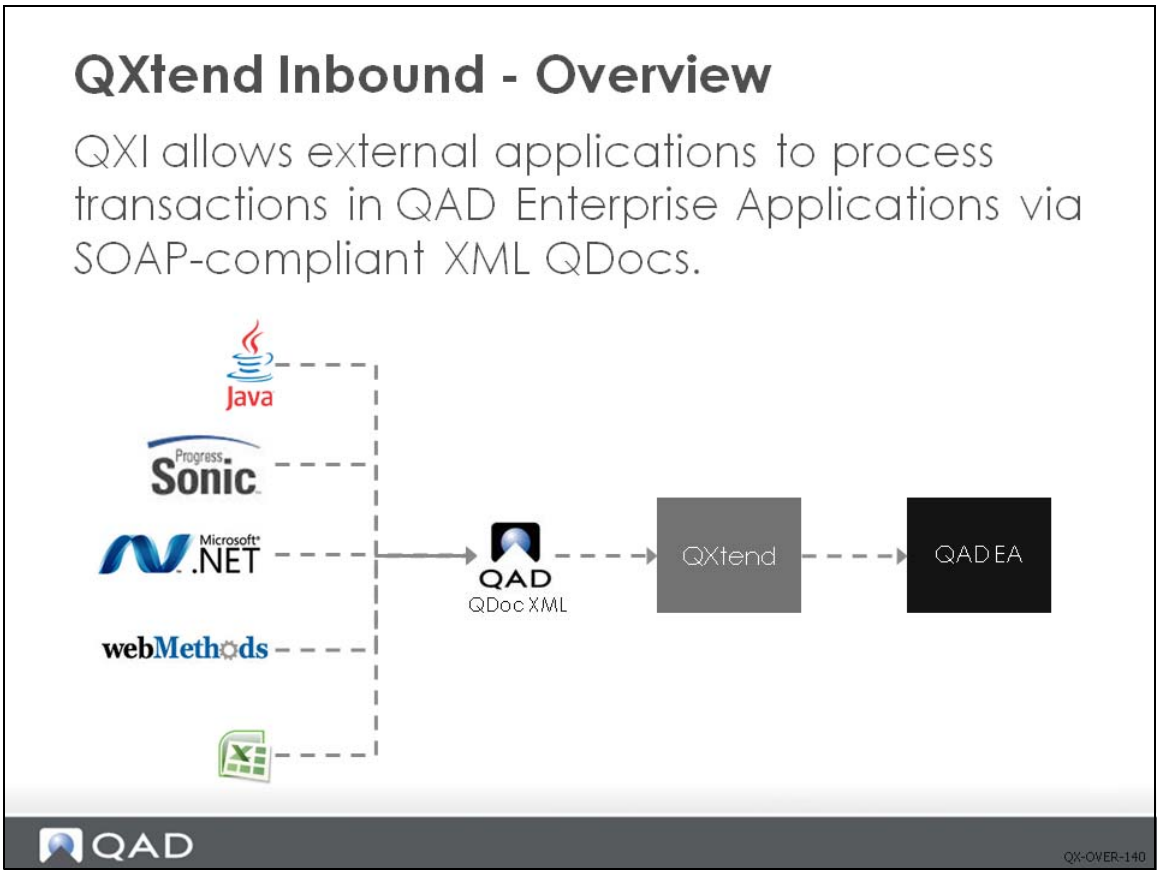
QAD QXtend Inbound



QXtend Inbound (QXI) enables external (non-QAD) applications to process transactions in QAD Enterprise Applications via SOAP Web Service messages. The contents of the SOAP messages have to follow the standards that are defined in the QDoc specification that is a QAD-controlled way of representing data that is processed by QXtend.

QDocs follow standards defined by the Web Services Interoperability Organization (WS-I) Following these standards allows QXtend to be used by any product that supports the WS-I standards. For more information on WS-I, see http://ws-i.org.

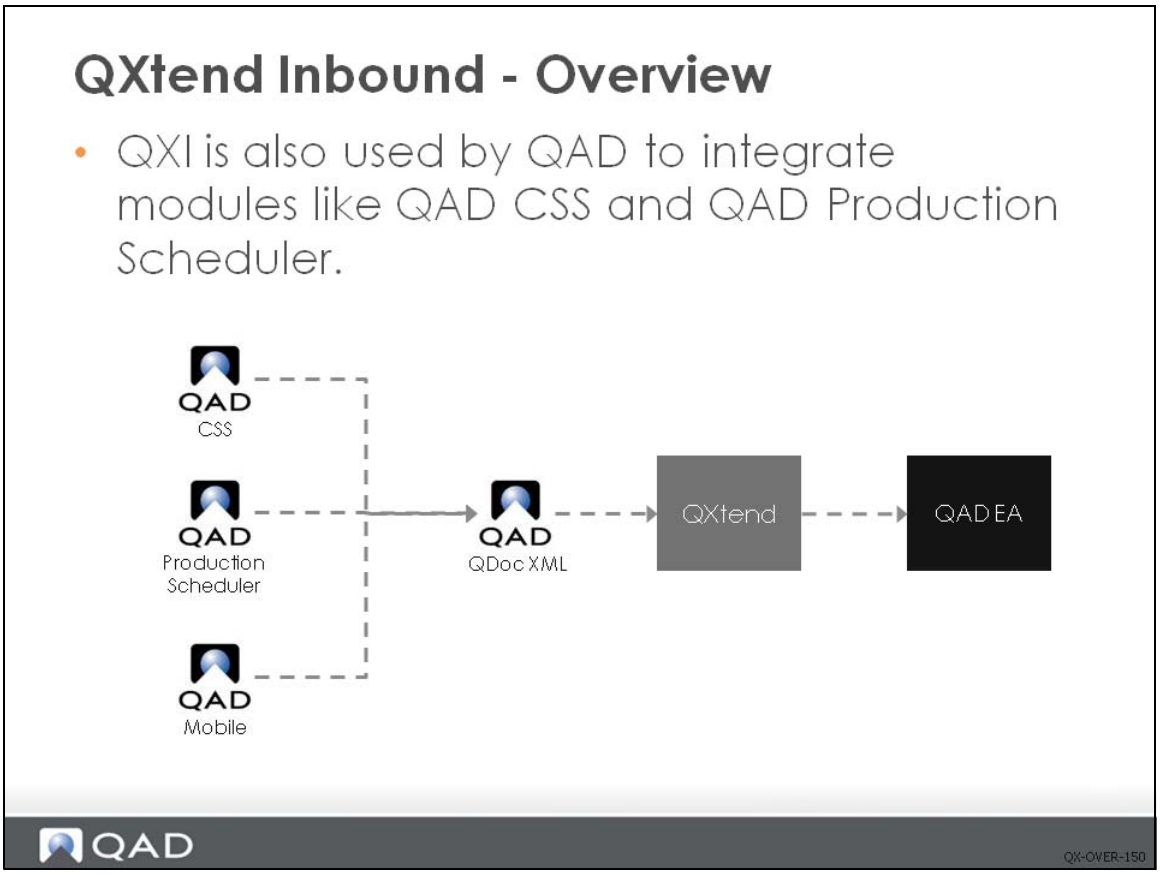
QXI is not only used to integrate external applications, but also to process transactions between QAD modules such as CSS and QPS.



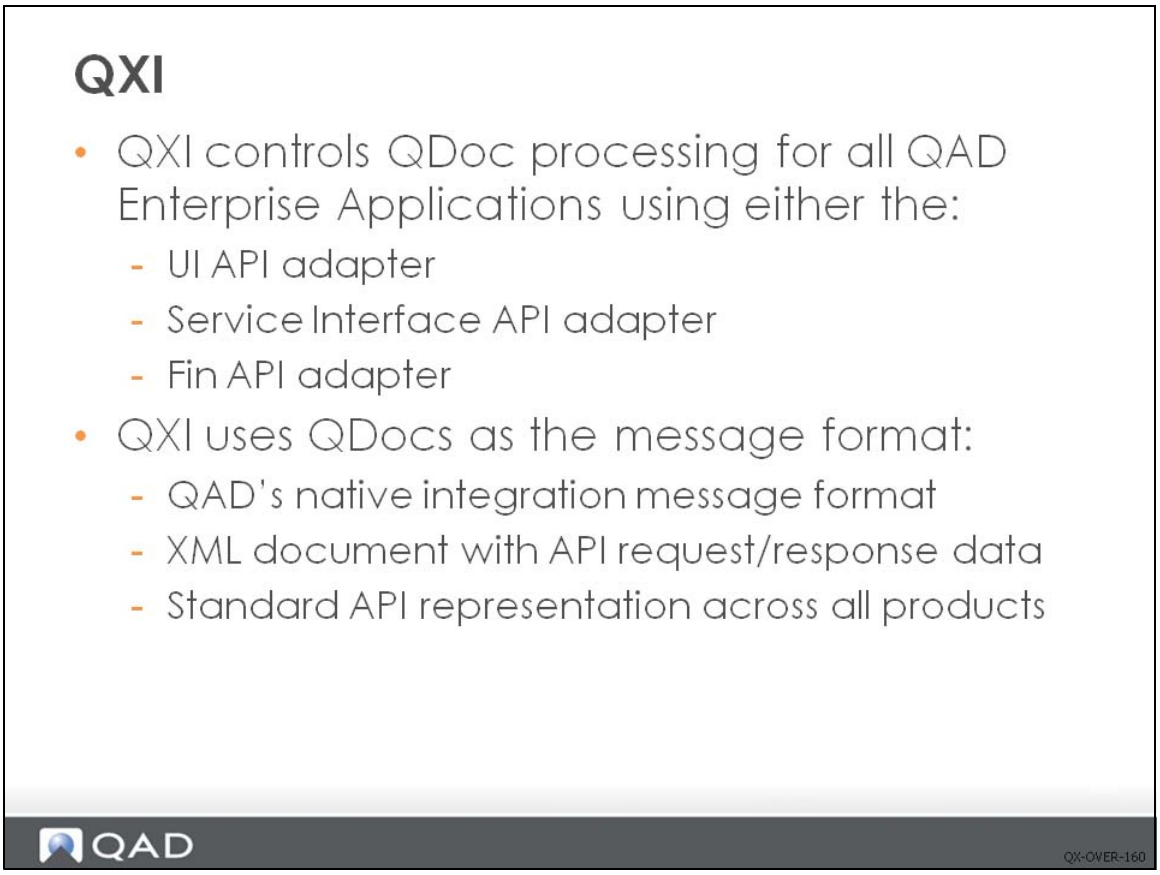
QXI provides a standard mechanism that can be used to load data into many of the QAD Enterprise Applications including the core ERP modules from the eB version onwards and from modules like QAD EAM, QAD CRM, and QAD PIM. Over time more of the standard integration for QAD modules will rely on the QAD QXtend toolset.

Loading data into the QAD Enterprise Application is enabled through a Web service that follows the guidelines established by WS-I. Ensuring that QXtend follows industry standards for Web services ensures that any application capable of invoking a Web service call load data into QAD Enterprise Applications. The slide above shows some application types that can use the QXI Web service. As you can see, anything from Microsoft to Java can leverage QXI.

Using a Web service ensures that most external applications can leverage the many APIs available through QXI. Web services also isolate the external application from the API implementation details; for example, the external application does not have to understand anything about Progress-based applications in order to communicate with one.



QXI has also been used by QAD to facilitate the integration of new modules developed by QAD. QAD modules like QAD CSS, QAD Production Scheduler, and QAD Mobile use QXtend's services to provide flexible module integration. Being able to use a common framework to provide integration solutions internally at QAD—as well as for external products—is one of the great value propositions of QXtend. A common integration framework simplifies the deployment and support of QAD module-to-module integration.



QXI controls the processing of QDocs for all QAD Enterprise Applications that provide APIs that are available to QXtend. Currently not all QAD modules support QXtend, but the modules that do will continue to grow. Eventually, QXtend will become the standard integration toolset at QAD.

QXtend Adapters

QDocs are processed using a QXtend adapter. QXtend currently supports three types of adapter: UI API adapter the service interface (SI) API adapter and Fin API adapter.

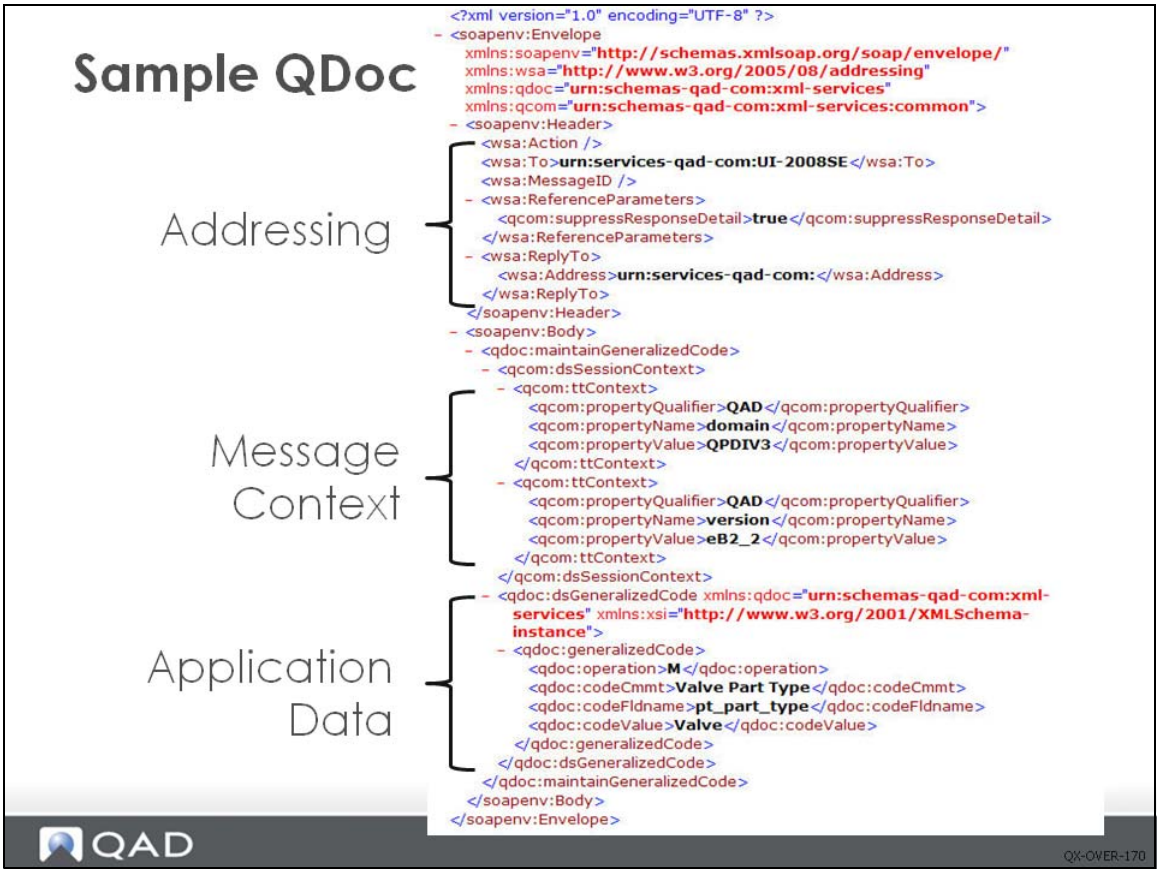
The UI API adapter provides access to most character screens developed for previous versions of QAD EA. The UI API adapter uses a telnet session to enter data from the QDoc being processed by QXI. The adapter loads the data using the character screen, and executes all business logic associated with that screen.

The SI API adapter enables the use of business logic implemented by the QAD reference architecture, which separates the user interface from the business logic. These APIs are invoked by using standard functions provided by the Progress AppServer. Because these APIs do not rely on the user interface, they are more efficient than the UI APIs.

The Fin API adapter is one kind of SI API adapter but is used for QAD Enterprise Financials. Fin API adapter is only available for QADEE, which includes QAD Enterprise Financials.

Requests and responses are passed to—and received from—QXI in a QDoc, which is a QAD-proprietary format XML document. QDocs include application data and any other data required to facilitate processing. QDocs define the interface to the application function and define a contract for the API.

QDocs




```
<?xml version="1.0" encoding="UTF-8" ?>
- <soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:qdoc="urn:schemas-qad-com:xml-services"
    xmlns:qcom="urn:schemas-qad-com:xml-services:common">
  - <soapenv:Header>
      <wsa:Action />
      <wsa:To>urn:services-qad-com:UI-2008SE</wsa:To>
      <wsa:MessageID />
    - <wsa:ReferenceParameters>
        <qcom:suppressResponseDetail>true</qcom:suppressResponseDetail>
      </wsa:ReferenceParameters>
    - <wsa:ReplyTo>
        <wsa:Address>urn:services-qad-com:</wsa:Address>
      </wsa:ReplyTo>
    </soapenv:Header>
  - <soapenv:Body>
    - <qdoc:maintainGeneralizedCode>
      - <qcom:dsSessionContext>
        - <qcom:ttContext>
            <qcom:propertyQualifier>QAD</qcom:propertyQualifier>
            <qcom:propertyName>domain</qcom:propertyName>
            <qcom:propertyValue>QPDIV3</qcom:propertyValue>
          </qcom:ttContext>
        - <qcom:ttContext>
            <qcom:propertyQualifier>QAD</qcom:propertyQualifier>
            <qcom:propertyName>version</qcom:propertyName>
            <qcom:propertyValue>eB2_2</qcom:propertyValue>
          </qcom:ttContext>
        </qcom:dsSessionContext>
      - <qdoc:dsGeneralizedCode xmlns:qdoc="urn:schemas-qad-com:xml-
          services" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
          instance">
        - <qdoc:generalizedCode>
            <qdoc:operation>M</qdoc:operation>
            <qdoc:codeCmmt>Valve Part Type</qdoc:codeCmmt>
            <qdoc:codeFldname>pt_part_type</qdoc:codeFldname>
            <qdoc:codeValue>Valve</qdoc:codeValue>
          </qdoc:generalizedCode>
        </qdoc:dsGeneralizedCode>
      </qdoc:maintainGeneralizedCode>
    </soapenv:Body>
</soapenv:Envelope>
```

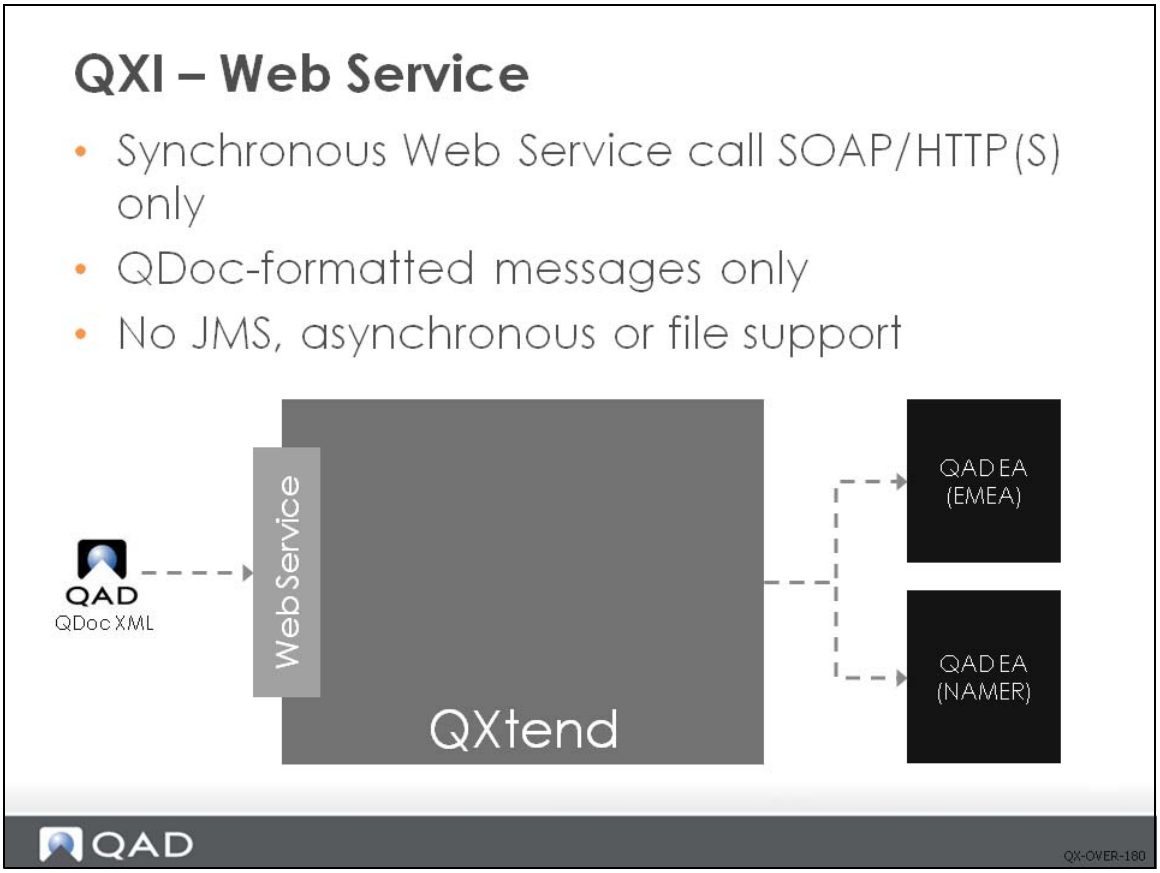
QXI loads data into QAD Enterprise Applications via Web service calls; every request and response is exchanged as a SOAP message. A sample QDoc SOAP Message processed by QXtend Inbound is shown in the picture above. A QDoc has three important areas:

Addressing. Describes the application that sent the data and the destination of the request. The destination is a receiver and identifies a specific instance of QAD EA that will receive the data in the request.

Message Context. Provides the context in which the message should be processed. This section contains information such as the QAD EA Domain and the actual version of the QDoc being processed.

Application Data. Contains the data that gets loaded into the application. The data here controls which data item gets updated and the data that gets loaded into the receiving application.

Web Service

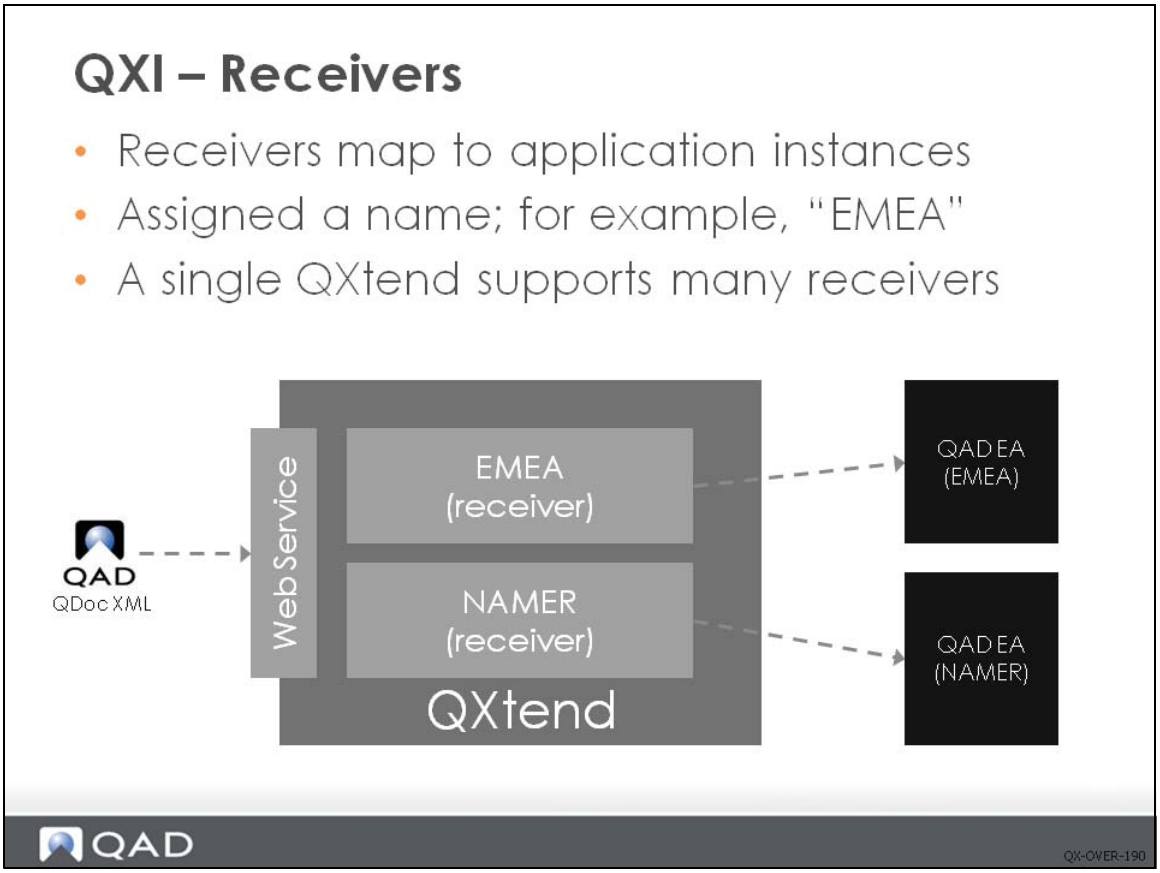


QXtend Inbound processes requests by passing correctly formatted QDoc SOAP messages over an HTTP connection. QXtend only supports the web service type SOAP/HTTP(s). SOAP is a standard way of formatting the request and has three components:

- SOAP envelope, which contains the message. This is like an envelope used to send a traditional letter.
- SOAP header, which contains the addressing information that identifies the sender and receiver of the message. The SOAP header is mandatory in the QDocs specification. This is like the address required when posting a traditional letter.
- SOAP body, which contains the data—customer, supplier, item, and so on—to load into the target application. To continue the metaphor of the traditional letter, the SOAP body is the letter included in the envelope.

QXtend does not natively support JMS, asynchronous messaging, or processing operating system files. Many different mechanisms could be used to get data into an application, and QXtend could not natively support every option—this is not where the value of QXtend is realized. Numerous products support many different integration methods, and QXtend has been designed to work seamlessly with these types of application. Sonic ESB, WebMethods, Web Sphere, Sun Jcaps, and Biz Talk are leading ESB/EAI tools that specialize in this area. All leading EAI/ESB tools can call Web services and therefore can work with QXtend. Many customers use QXtend successfully with several EAI/ESB tools. QAD are working on introducing Sonic as a standard component to complement QXtend.

Receivers



QXtend provides a centralized integration hub that is not tied to any specific QAD Enterprise Applications version. The receiver in QXI identifies the QAD Enterprise Application instance; the instance could be core modules or add-on modules that use QXtend, such as QAD EAM and QAD CRM.

The slide above shows two receivers; each receiver is a different instance of the core QAD Enterprise Application suite. One instance serves North America, the other serves EMEA. These are two entirely separate instances of the software, but both are connected through a single QXtend instance. The versions of the NAMER and EMEA do not have to be identical as QXtend supports many versions.

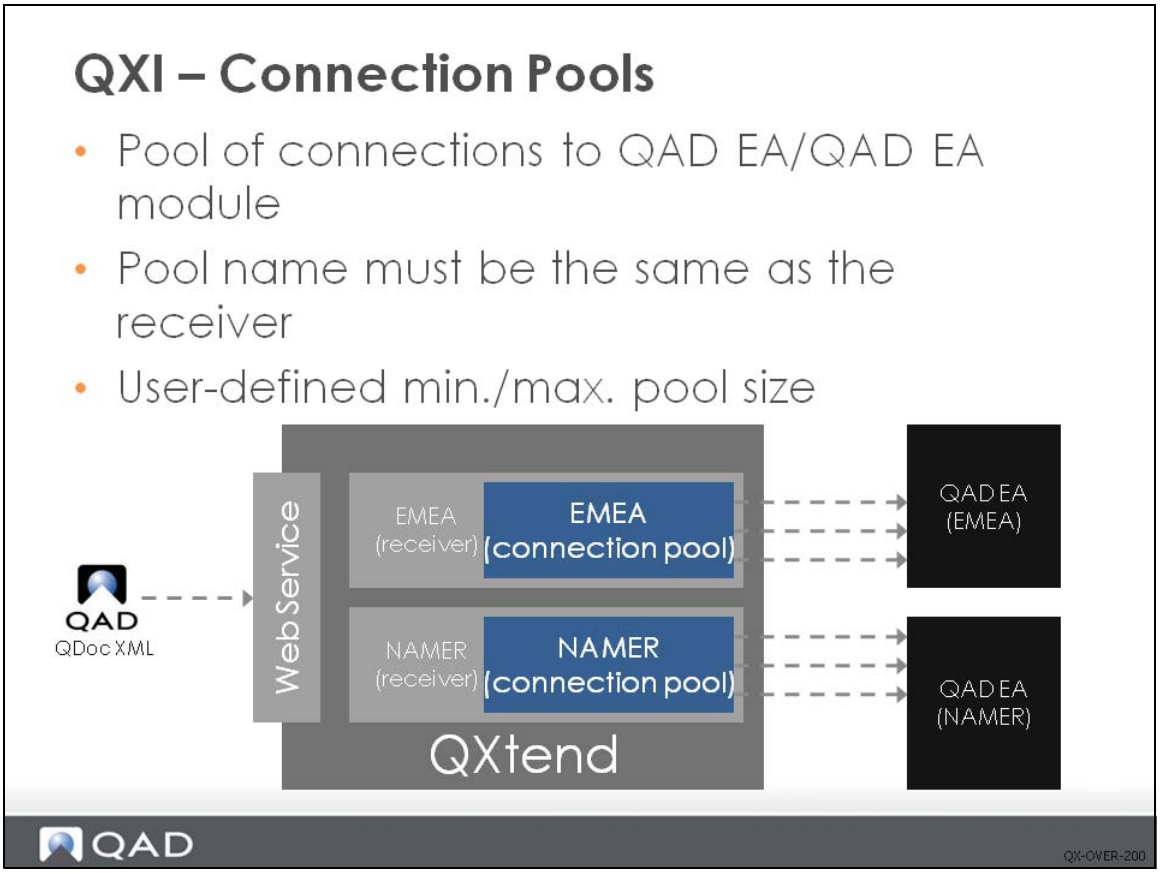
The name of the receiver is a critical piece of information that must be supplied in any request that is processed. If the receiver is not in the request—or if the receiver is invalid—the message is not processed and an exception is returned to the caller as a SOAP fault.

If a request is intended for the NAMER receiver, the following must be included in the SOAP header:

```
<wsa:To>NAMER</wsa:To>
```

Note The receiver name in the request QDoc is case-sensitive, so the name in the request and the name of the receiver in QXtend must be identical.

Connection Pools



QXI provides an easily managed multi-threaded environment, meaning that multiple requests can be processed simultaneously. CIM and other solutions that are Progress-based do not offer native multi-threading and require many processes to be managed manually. Through the connection pool concept, QXtend can provide a easily managed layer that facilitates the processing of simultaneous requests.

Receivers must have at least one connection pool associated with them, but can have multiple connection pools of different types. Connection pools are assigned a name that must match the name of the receiver that the connection pool is for. Since QXI is case-sensitive, the case of the name must also match.

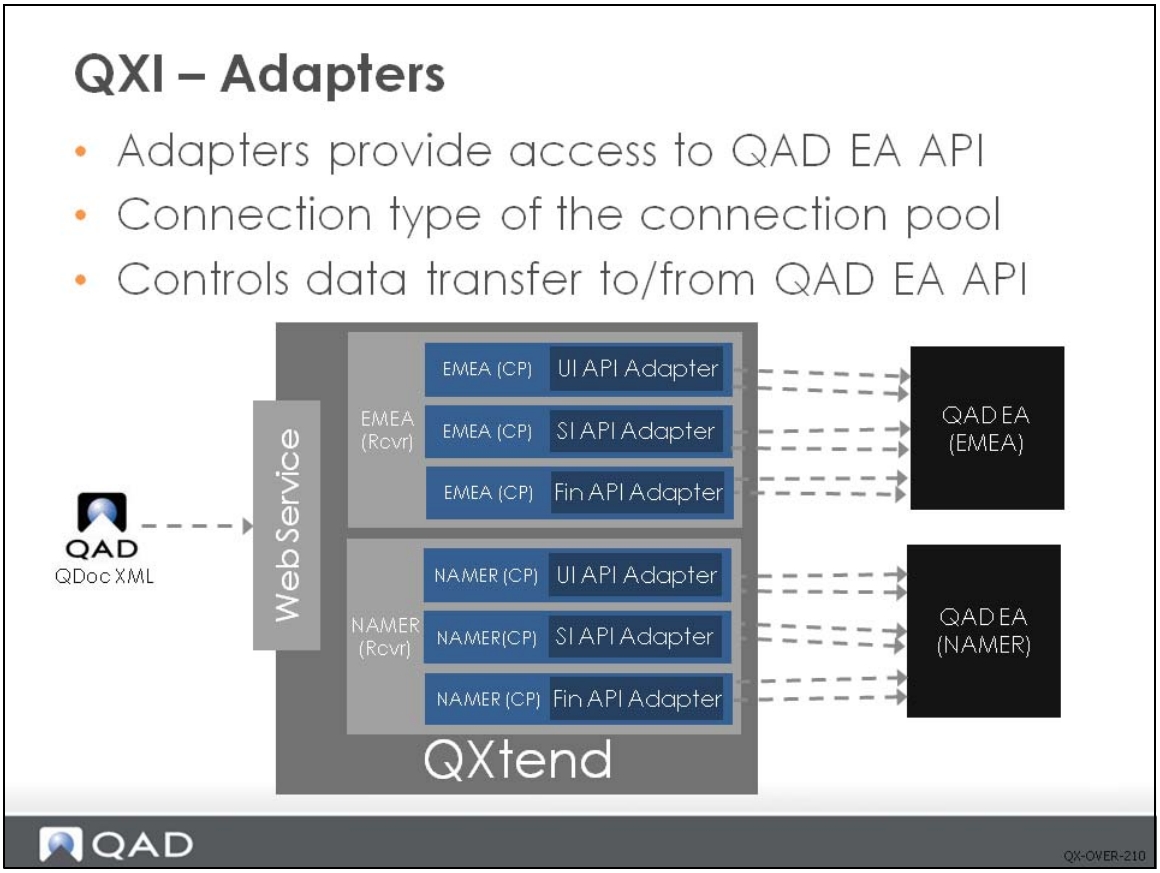
The Connection Pool holds a pool of sessions connected to an instance of QAD Enterprise Applications. These connections are used to pass data from QXI to the QAD EA instance. When a connection pool is created, connection parameters must be configured to enable the connection between QXI and QAD EA to be established. These parameters include:

- User Name
- Password
- Instance Location (host, port, and so on)
- Minimum Pool Size
- Maximum Pool Size

Connection parameters control the behavior of the connection pool. The Minimum Pool Size and Maximum Pool Size control the multi-threaded processing of QXtend. When QXI starts, it builds the connection pool and creates the minimum number of connections into QAD EA. As requests are processed, QXI tries to keep the minimum number of connections available and increases the connections available in the pool until it reaches the maximum pool size.

The connection pool controls the number of requests that can be processed simultaneously into a QAD EA instance.

Adapters



Adapters are another fundamental component of QXI. An adapter controls access to the APIs in QAD EA. The way in which an API is implemented affects how the API is accessed. As technology changes, new ways of writing APIs evolve, some of which are incompatible with previous approaches. The use of adapters in QXI allows QXtend to shield users from the effects of changes in technology: as a new method is implemented by QAD, QXtend simply provides a new adapter for the new method. This provides value to customers as they can continue to use a consistent approach; QXtend handles the calling of the APIs, thus abstracting the API implementation from the calling application.

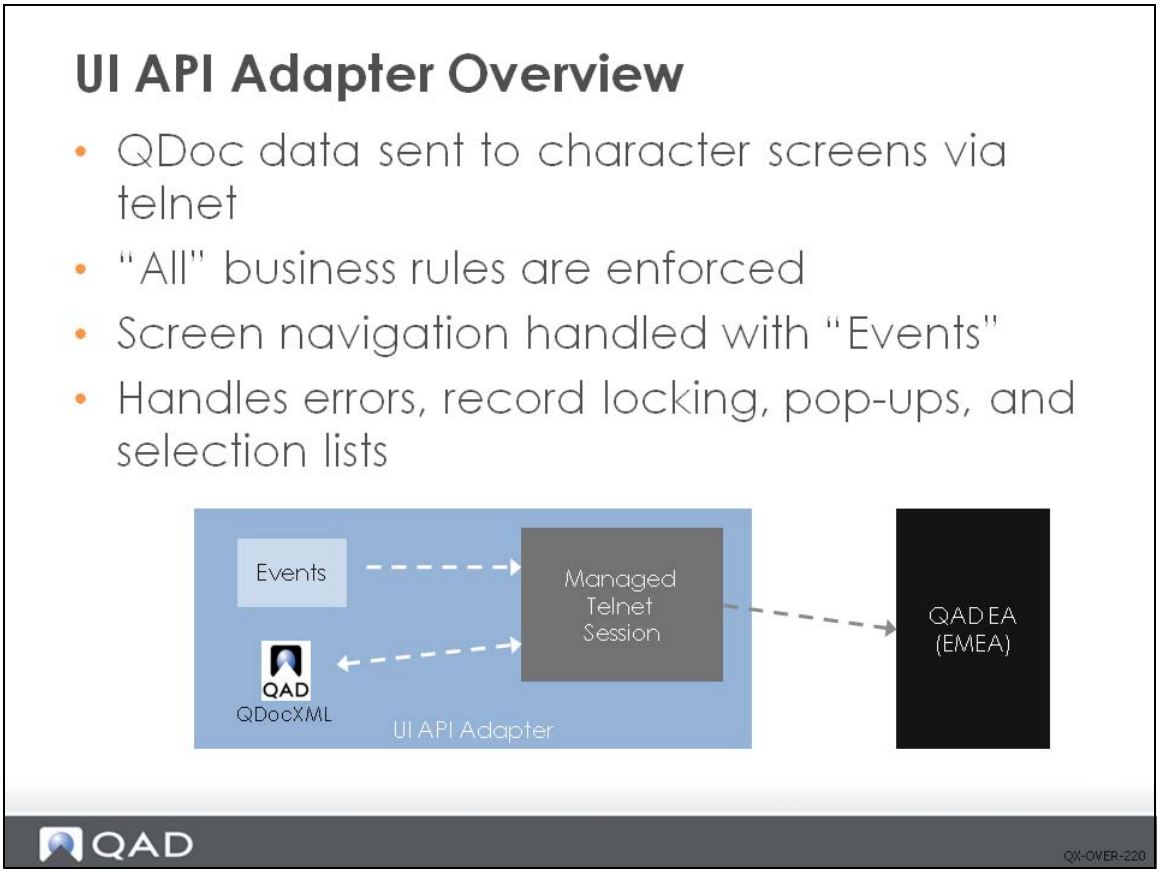
As described, a receiver can have one or more connection pools; each connection pool has a type. The type of the connection pool is linked to the adapter. As shown above, the EMEA receiver has three connection pools: one for the UI API adapter, one for the SI API adapter and the other one for Fin API adapter. This allows receivers to process requests to different types of API and the API type is not important to the caller they are just calling QXtend.

The adapter concept has already proved invaluable to QAD. QXtend currently supports three primary adapters: the UI API adapter, the SI API adapter and the Fin API adapter. When first released, QXtend supported the UI API adapter primarily, which uses a telnet connection to the character screens and loads data by simulating the actions of a user. However, the performance of the UI API adapter does not meet some high volume requirements placed on applications today.

The SI API adapter was introduced into QXtend in the 1.4 release and provides a mechanism for executing Progress business logic that resides on a Progress AppServer. The SI adapter does not rely on the user interface to enter data. The SI API supports high volumes and is also supported by EAM, and CRM modules. Using adapters in QXtend has protected customers from the major changes that have been implemented to QAD EA.

The Fin API adapter is one kind of SI API adapter but is supported by the Enterprise Financials

User Interface Adapter



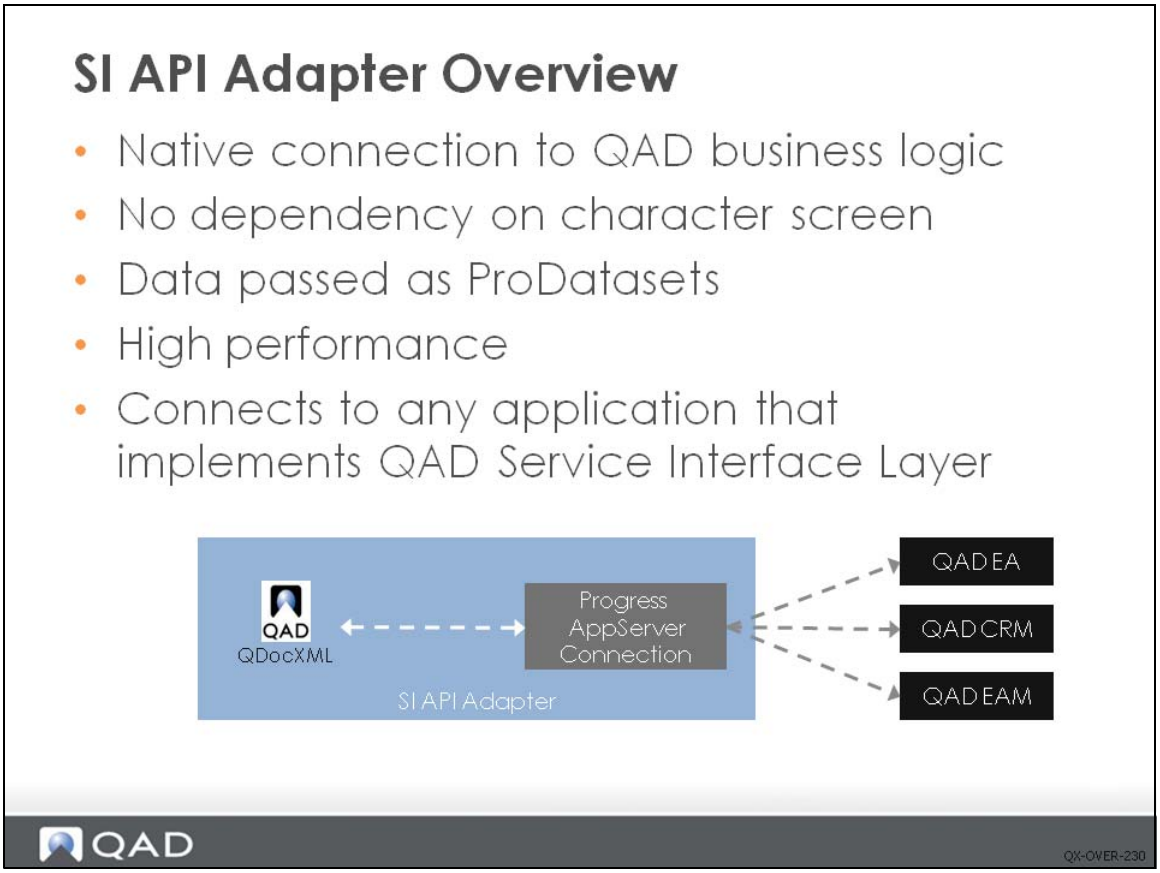
The UI API adapter covers almost all core QAD Enterprise Applications by providing a mechanism that loads data through the character screens that a user usually employs to enter data into the system. This approach does not require a change to the implementation model used and works with all versions of QAD Enterprise Applications from eB onwards. The adapter performs the actions that a user would usually perform to load data into the applications. As such, all business rules that are normally enforced—including generalized code validation—are also enforced as data is loaded via the UI API adapter.

Data is sent from the UI API adapter to the character screen field by field; the current position in the QDoc XML request is maintained by the adapter. If an application error occurs as data from the QDoc is entered into the application, the error is trapped immediately and mapped back to the exact field in the QDoc that caused the error. That field, in turn, is mapped back to the exact location in the QDoc request XML. A response QDoc is generated that contains the details of the exception and the context of the error, which help identify the exact location of the problematic data.

The application character screen drives the integration requesting the data from QXI, and therefore when new screens or pop-up windows appear, QXtend can handle these screens and will not break when a previously unseen frame is encountered. The data pull approach provides a more robust mechanism than the error prone CIM alternative, as well as an error handling/reporting and record lock processing.

A major component of the UI API adapter is the event mechanism. As described previously, the adapter mimics the actions of a user. Some application screens have complex interactions and do not always follow the same rules. For example, usually the F4 key is used to back out; however, sometimes F4 means continue. The event mechanism enables the adapter to cope with these variations by storing the necessary navigation actions that are required to navigate successfully through the user interface of the function. The events are created using the QGen tool to map the screen and record the events; the events are then used when processing a request through the UI API adapter.

Service Interface (SI) Adapter



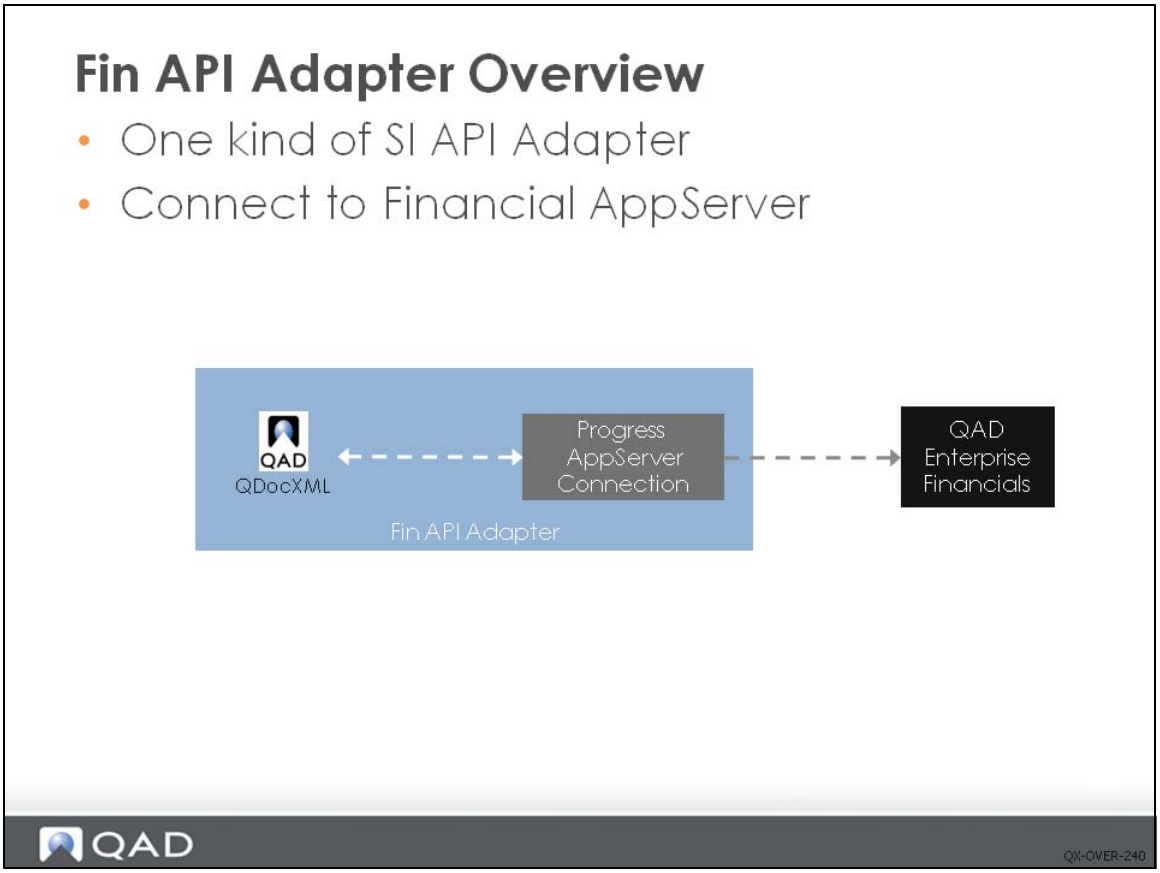
The SI adapter provides an adapter to Progress-based business services (APIs) that have followed the QAD Service Interface implementation guidelines. The service interface provides a set of infrastructure components that allow QAD to follow a standard pattern using ProDatasets from Progress OpenEdge. QAD modules that support the service interface can be called easily from QXtend without the need for new adapters to be created; this also enables other modules to move their APIs to the service interface and then also have those APIs available through QXtend.

The SI adapter is different from the UI API adapter as it does not use character screens. Instead APIs are structured as pure business logic and do not execute any UI logic as part of the API. Removing the dependency on the character screens allows the SI adapter to outperform the UI API adapter. Data is sent to the API in one message rather than field by field, all business rules are executed, and a response is returned to QXI. In turn, QXI builds a response QDoc that is sent as the response to the Web Service call.

QAD does not provide all the APIs since this would require existing application logic to be restructured to separate the user interface and the business logic. This restructuring is a time-consuming process; consequently using the UI API adapter allows QAD application logic to be restructured in a phased approach, starting with QAD 2008 EE, which makes the APIs available through the QXtend SI API Adapter.

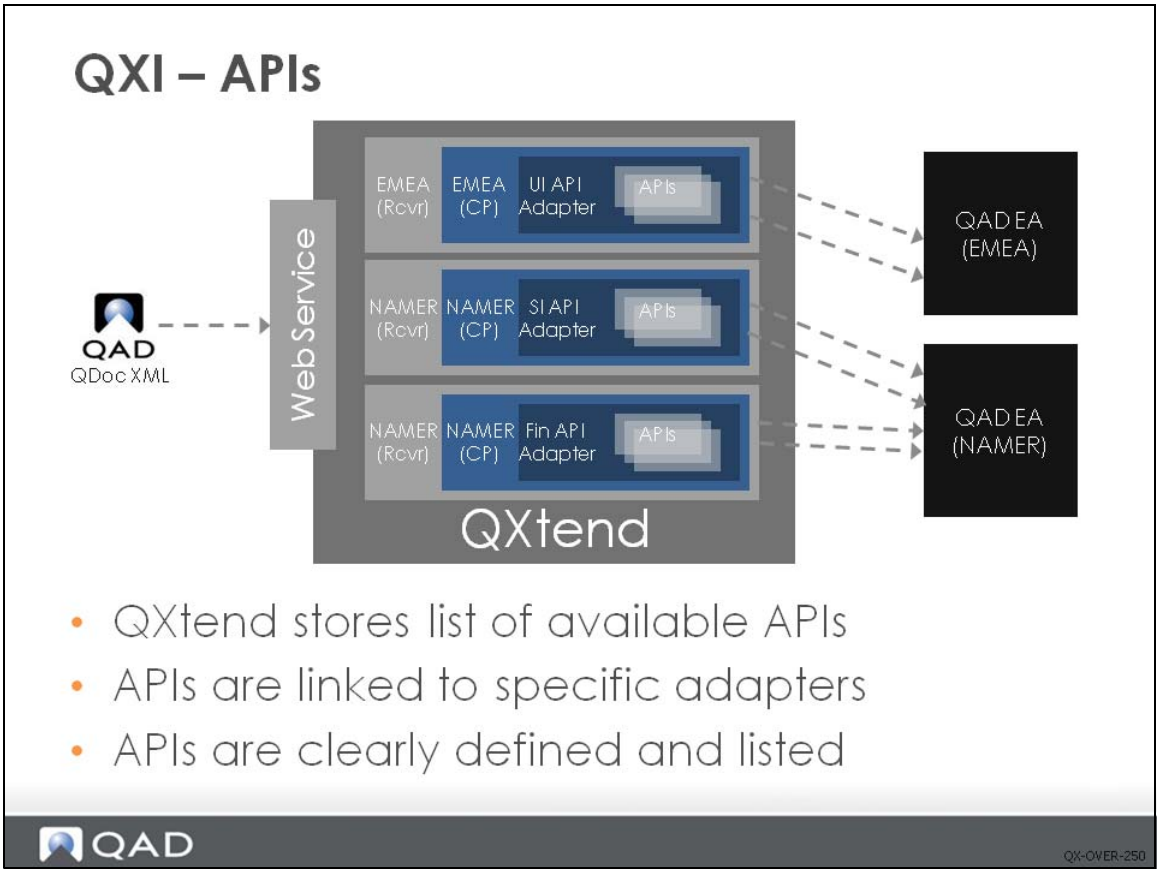
Other QAD modules such as QAD EAM and QAD CRM also implement the service interface.

Fin API Adapter



Fin API Adapter is one kind of SI API Adapter which is used by QAD Enterprise Financials. When connecting to QAD EE, it will require creating two separated SI API Adapters to connect to qadsi and qadfin AppServers. QXtend cannot have two SIAPI connection pools using the same pool name. So Fin API Adapter was introduced into QXtend in 1.6 release to connect to Financial AppServer.

APIs

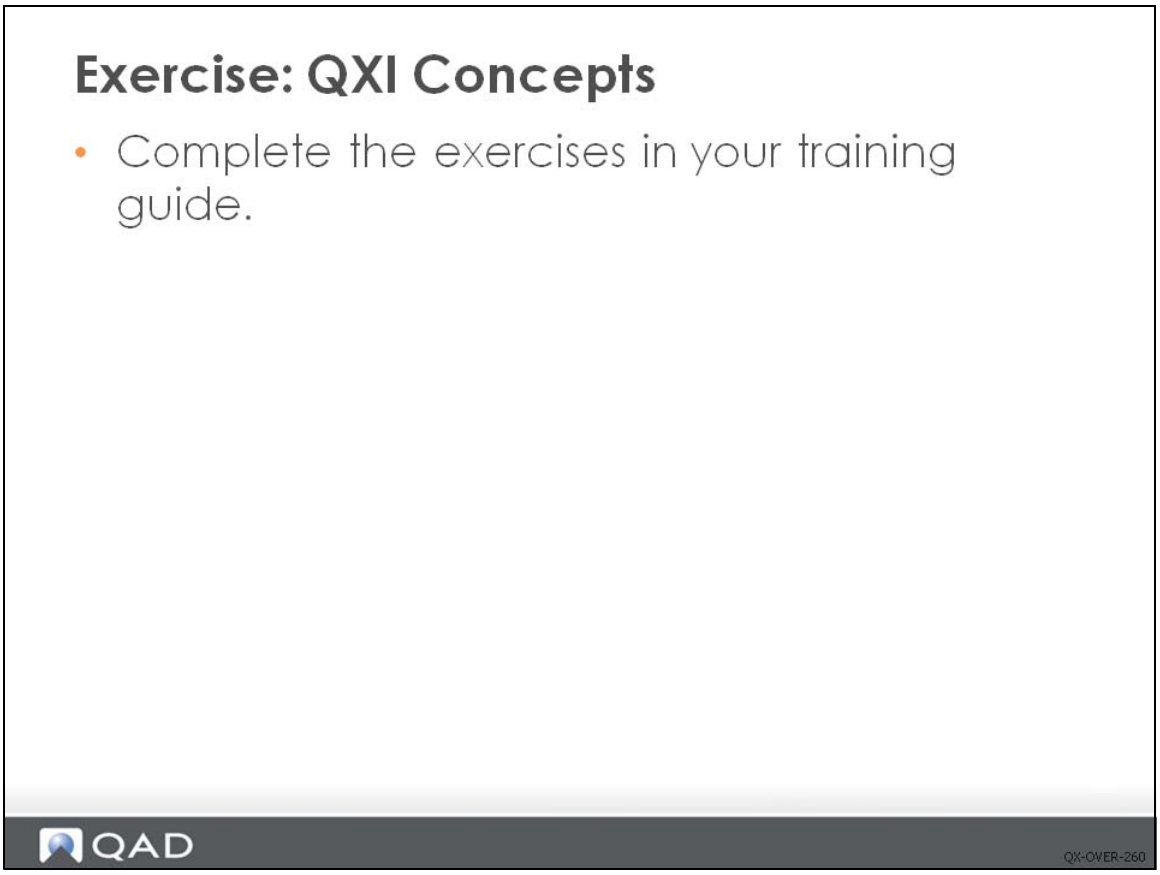


QXI provides access to many APIs; the list of available APIs is maintained in QXtend under the Schemas section. The schemas define the API. An API has two schemas, one to define the content of request message, the other to define the content of the response message. QXtend can create WSDL documents for selected receiver APIs that describes the SOAP request and response by referencing the schema definitions.

API names have two components <*apiName*>-<*apiVersion*>, the combination of these create a unique identifier for the API. A specific version of an API is linked directly to an adapter; QXtend maintains the list of APIs that are valid for an adapter.

Example imaintainItem-ERP3_2 is linked to UI API Adapter then when an maintainItem SI API is created the new API would be called maintainItem-ERP3_3.

A receiver can access all APIs available in QXtend. However, during the configuration of QXI, the API's supported by the receiver must be selected from the list of available APIs. This feature allows the APIs that are accessible through QXtend to be restricted on a receiver-by-receiver basis.



The following list shows a number of key terms and concepts used in QXI. In each statement below, fill in the correct term from the list. Some terms may be used more than once.

Web Services Interoperability	WSDL documents
UI API adapter	SOAP/HTTP(s)
application data	SI API adapter
SOAP header	QDoc
Fin API adapter	case-sensitive
requests	connection pool
addressing	Web service
receiver	SI adapter
message context	SOAP body

1 QDocs follow standards defined by the _____ organization.

2 Loading data into the QAD Enterprise Application is performed via a _____.

3 QXtend currently supports the _____,the _____ and the _____.

4 The _____ provides access to most character screens.

5 The _____ enables the use of business logic implemented by the QAD reference architecture.

6 The _____ is the only API that provides access to QAD Financials.

7 Requests and responses are passed to—and received from—QXI in a _____, which is a QAD-proprietary format XML document.

8 A QDoc has three important sections; these are the ___, _____, and _____ sections.

9 QXtend only supports the web service type _____.

10 The data to be loaded into the target application is contained within the _____.

11 It is the _____ that contains the addressing information that identifies the sender and receiver of the message.

12 The _____ in QXI identifies the QAD Enterprise Application instance.

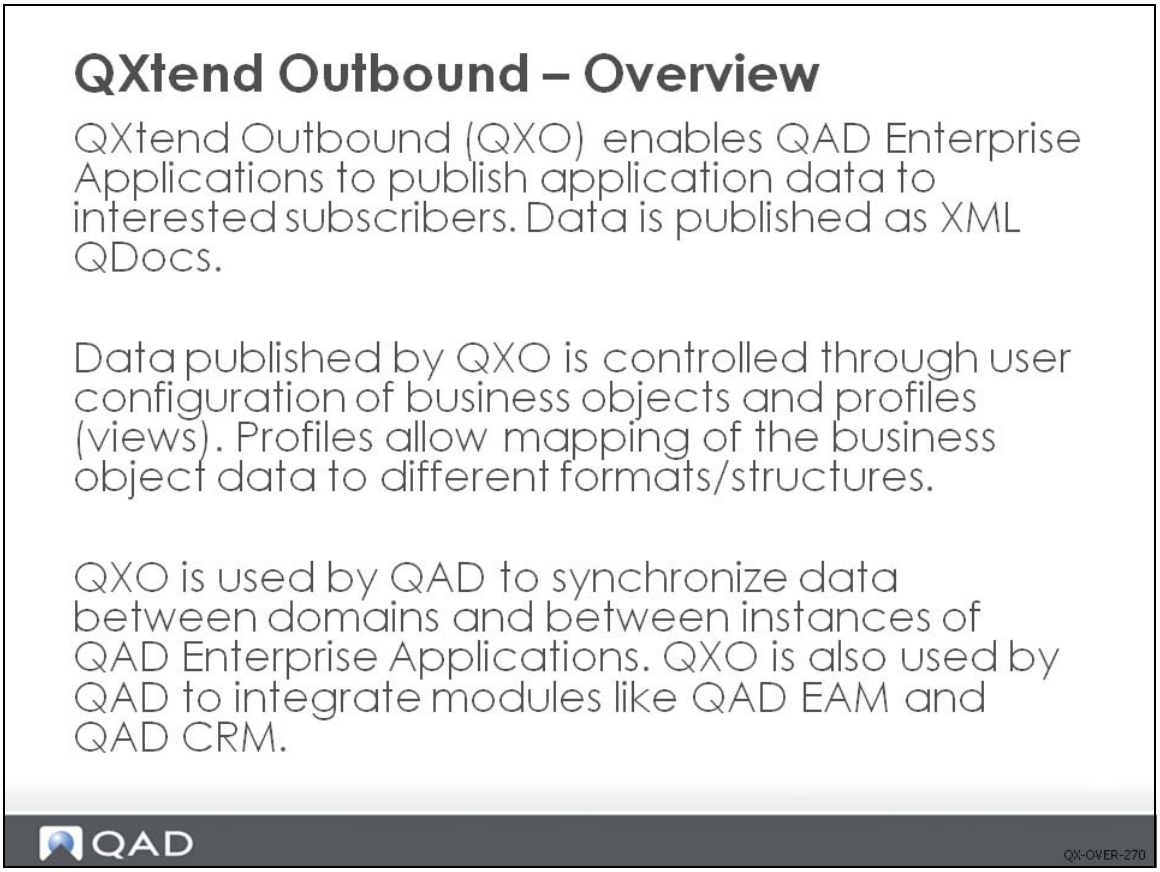
13 The receiver name in the request QDoc is _____. Consequently the name in the request and the name of the receiver in QXtend must be identical.

14 QXtend facilitates the processing of simultaneous requests via the concept of the _____.

15 The connection pool controls the number of _____ that can be processed simultaneously into a QAD EA instance.

16 QXtend can create _____ for selected APIs that describes the SOAP request and response by referencing the schema definitions.

QAD QXtend Outbound

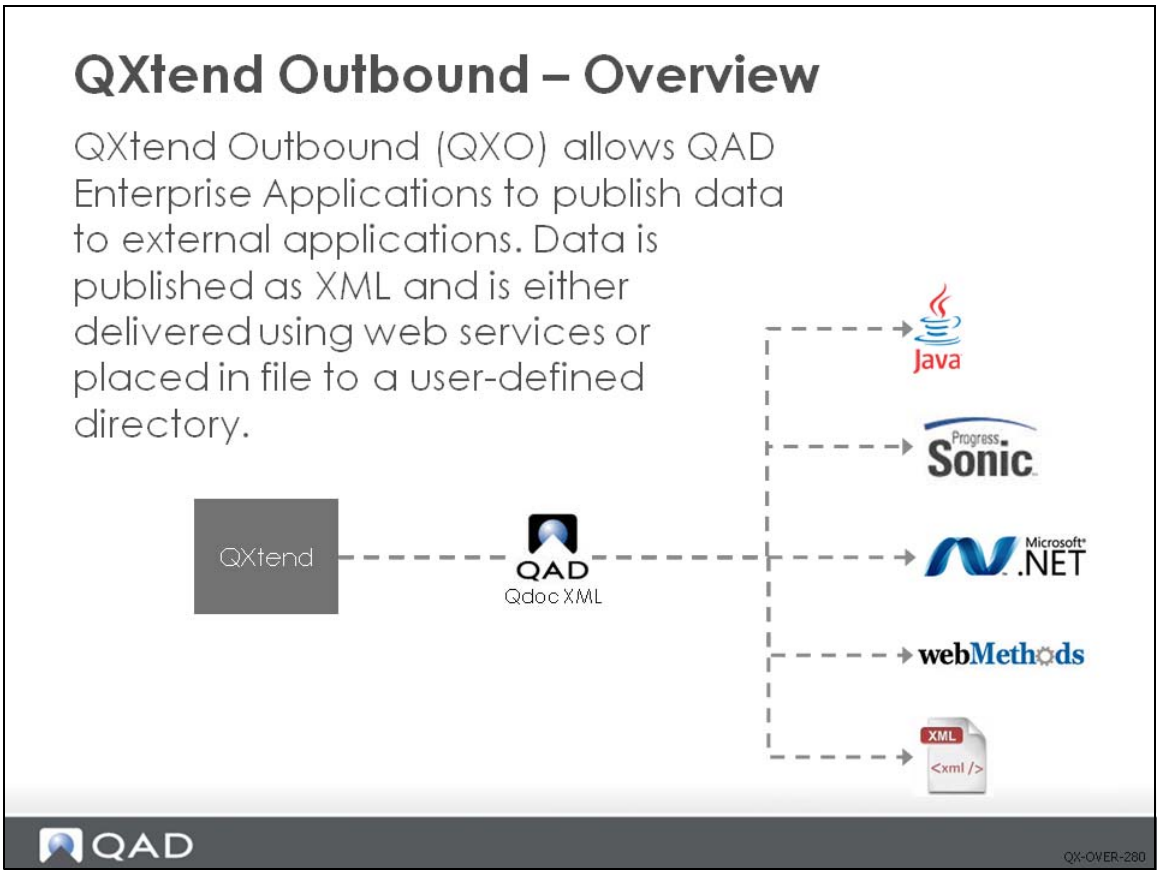


QXtend Outbound (QXO) provides a flexible, generic mechanism for publishing data from QAD Enterprise Applications (QAD EA). It takes many applications to run an enterprise, and the ability to share data between applications quickly and efficiently is critical.

QXO publishes QAD QA data as changes occur. QXO enables a change to be published to one or many subscribers. In this way as data changes the other applications in the enterprise are sent those changes automatically.

QXO provides tools that enable the user to control the data published by QXO in order to define the content of the data sent to subscribing applications. Business objects hold the raw data published/extracted from QAD EA. Profiles in QXO provide a user-defined view of the business object and allow the user to select which data from the business object to include. Profiles also allow the XML element names in the QDocs to be changed to meet the requirements of the external application. Profiles are used to change the format of the outbound QDoc into the inbound QDoc format when synchronizing data between domains or instances of QAD EA.

QXtend is often used to synchronize master data maintained in a master domain—this process is driven by QXO. QXO tracks the changes to data in the master domain and then controls the process of updating each of the subscriber domains or QAD EA instances. The same principles are also used to ensure that changes in data are reflected in the QAD EAM and QAD CRM solutions. The master instance is the source of the data for business objects, and profiles are used to map the data to the formats required by other domains and QAD modules. The data is then delivered to the individual domains or instances using the subscription mechanism in QXO.

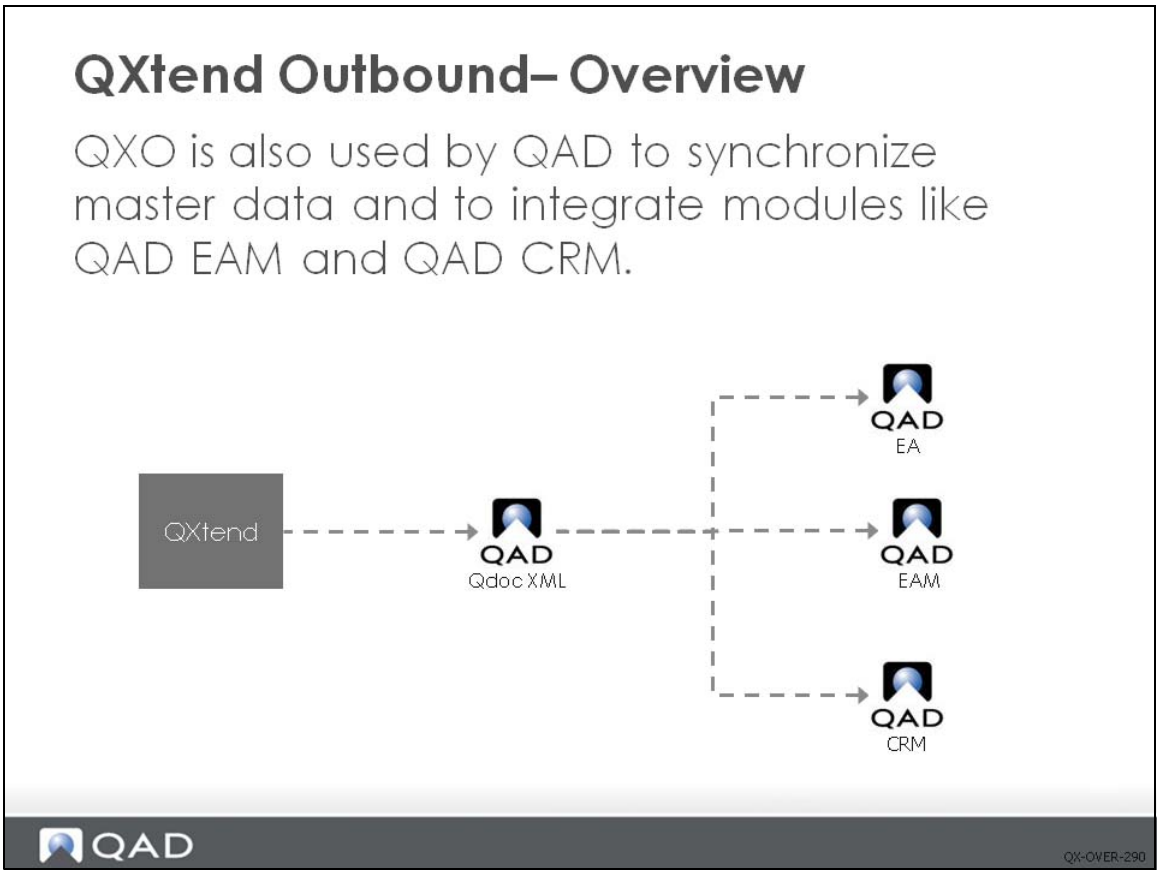


QXO provides a flexible engine for communicating changes to application data across the business enterprise. The continual changes to QAD EA data are listened to by QXtend and then published to interested applications in an appropriate format and structure. The data changes that are listened to in the QAD EA by QXO are controlled from within QXtend.

QXO supports two mechanisms for delivering messages to interested subscribers: web service call or file drop. The two delivery methods are configurable per subscriber.

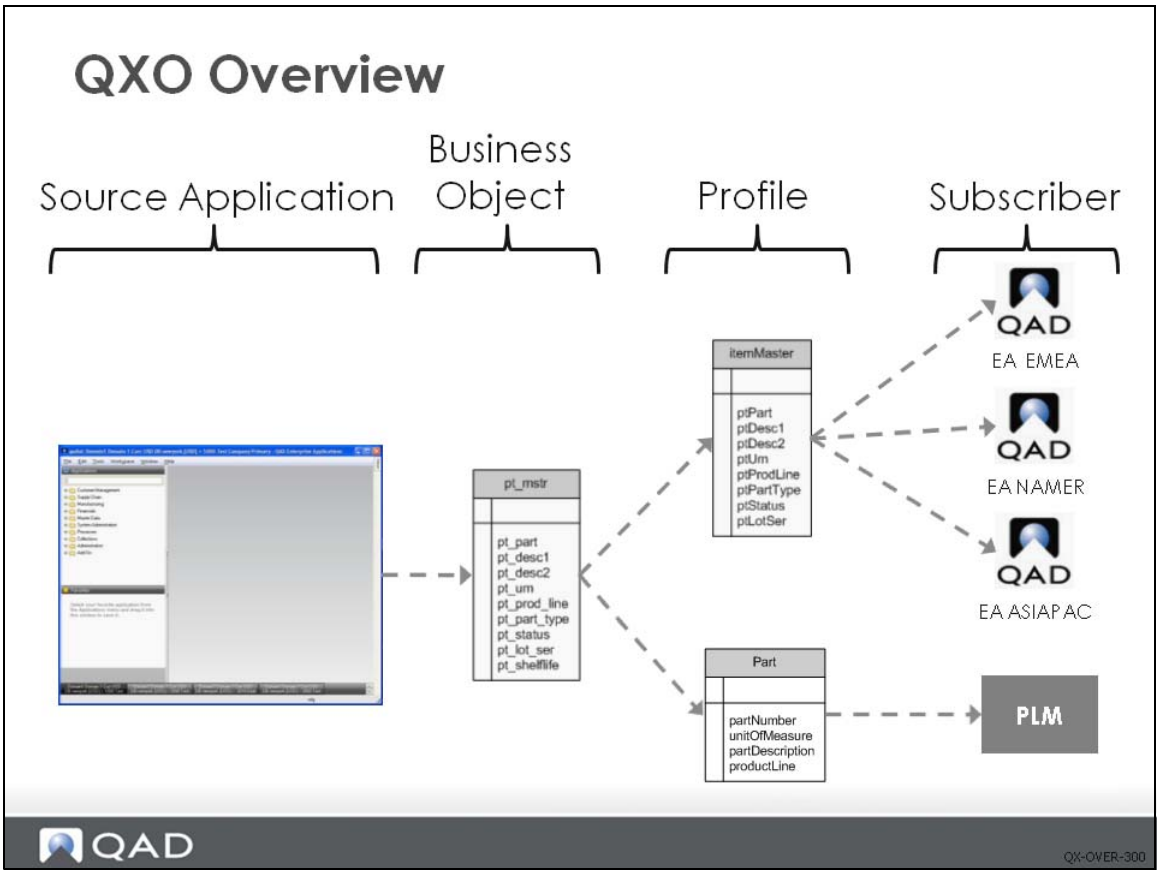
Web service delivery provides the capability to call QXI to chain QXI and QXO together—this is how data synchronization is achieved with QXtend. Web service delivery also provides a way to chain processes together within QAD EA. When calling the QXI web service, much of the configuration for the web service call is hidden from the user; only information required to successfully process in QXI is required. If calling a service other than QXI, more configuration is required as well as knowledge of SOAP messaging.

One of the fundamental drivers for QXtend is to provide ways for QAD EA data to be integrated with standard middleware/EAI products. All leading middleware solutions provide mechanisms to get data into the middleware environment, including http and file directory polling. The web service support and file drop mechanisms enable QXO data to be integrated into middleware/EAI solutions and, as such, provide QAD applications with an open architecture.



The QXO engine has also been used by QAD to drive the synchronization of master data (items, customers, suppliers, BOMs, and so on) between domains and between instances of QAD EA. Data synchronization used to be performed by the DataSync and Q/Linq products but since the release of QAD Enterprise Applications 2008 both of these products have been deprecated and replaced with QXtend.

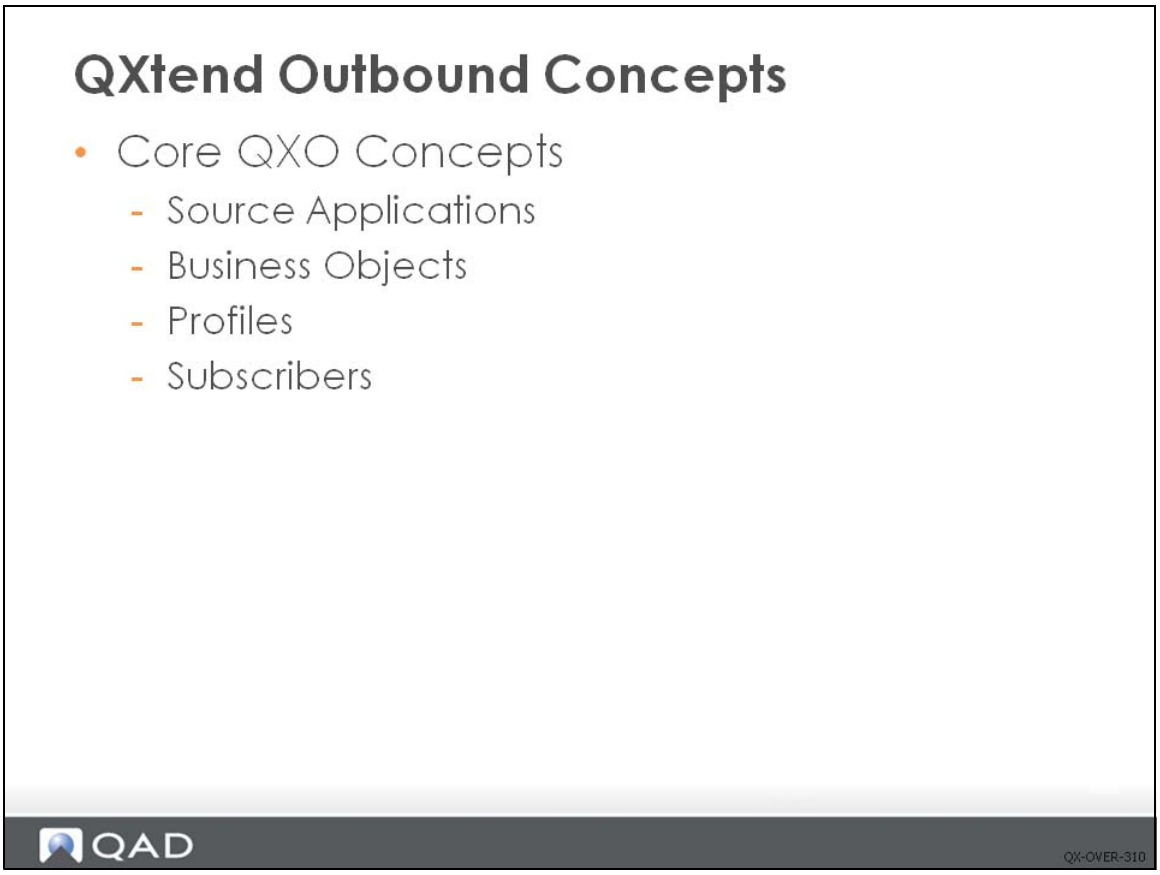
QXO is also critical component in the integration of modules such as QAD EAM and QAD CRM with the core modules of QAD Enterprise Applications.



The diagram above shows how data moves from the data source (source application) through QXO to the final destination (subscribers).

Data from the source application is first moved to a business object, which is a subset of the data stored in the source application. Business objects can be user-defined or fixed (defined by source application). Once the data has been stored as a business object in QXO, it is mapped to the profiles (views) that have been defined against it. The data in the profiles is then delivered to the subscribers that have registered an interest in the profile data.

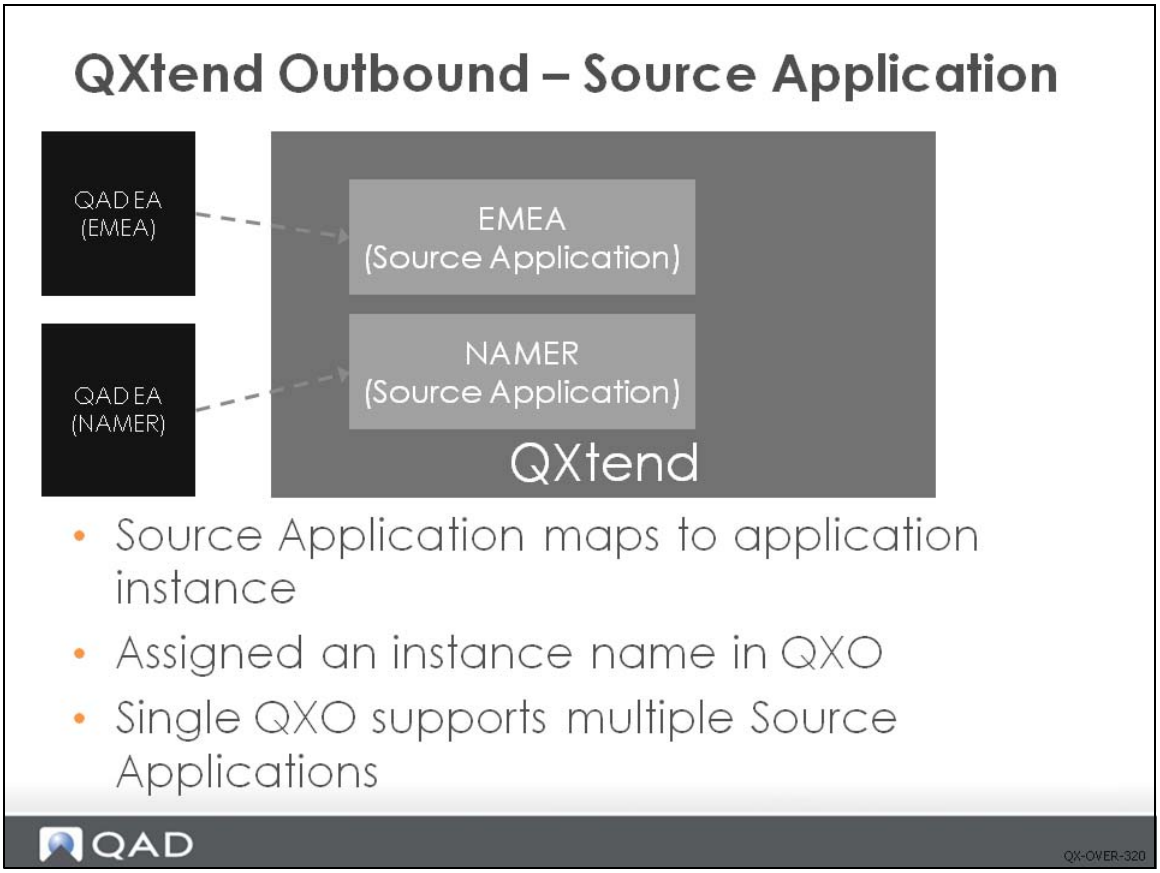
This flow of data is driven by services that move the data step to step in QXO. The fundamental process in QXO is to make data from one or more source applications available to any other application in the enterprise without having to tightly couple the two applications.



Several core concepts form the foundation of QXO; a good understanding of these is essential:

- Source applications
- Business objects
- Profiles
- Subscribers

Source Applications



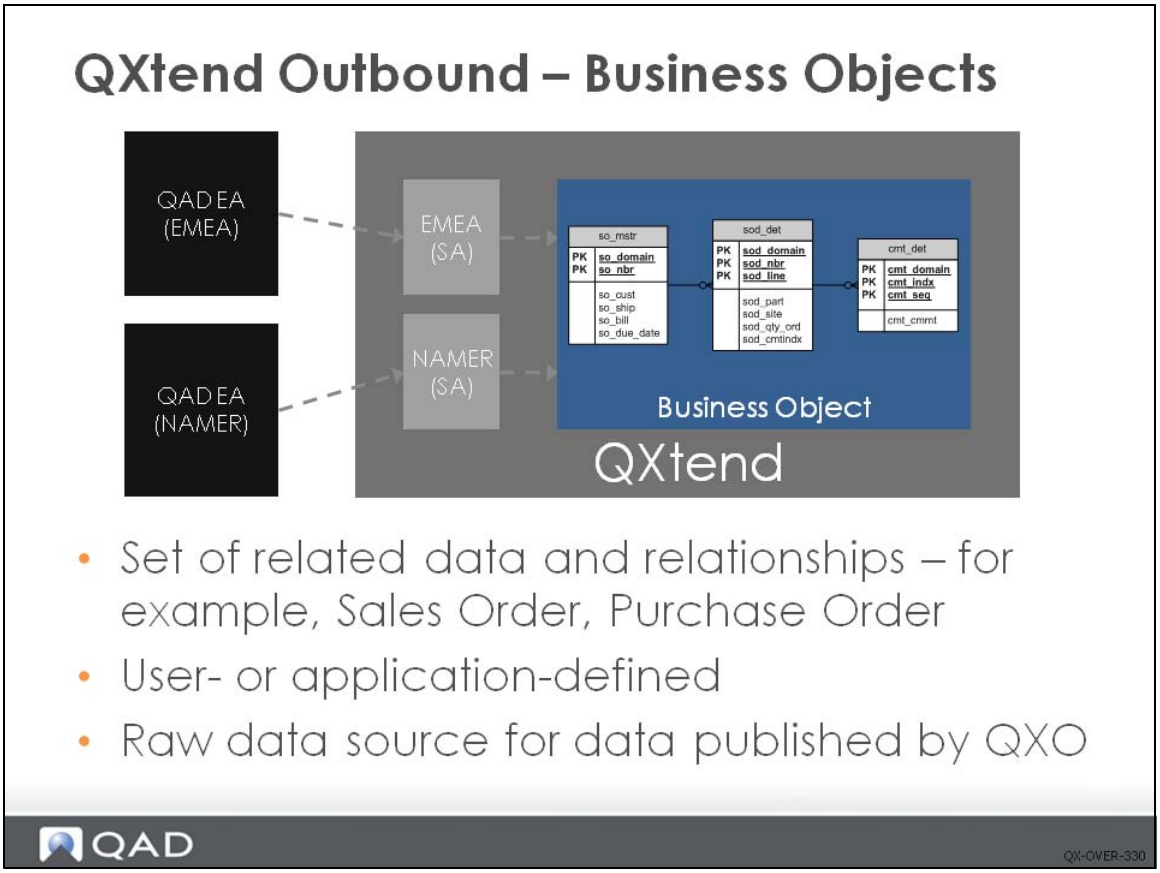
QXO provides a configurable, flexible, and non-invasive mechanism to distribute data from any instance of a QAD Enterprise Application to other applications in the business enterprise, as and when business events occur.

The source application in QXO is the source for the business data that is distributed. A single installation/instance of any QAD Enterprise Application maps to a single source application in QXO. The source application is assigned a name that is used to identify it. The source application in QXO is analogous to the receiver in QXI as both are used to identify an instance of QAD EA.

A single instance of QXtend supports multiple instances of QAD EA, including modules such as QAD EAM and QAD CRM that are additional data sources to core QAD EA. The source application is the connection to a single QAD EA instance, QXO allows the creation of multiple source applications, and each points to a different QAD EA instance. Configuring multiple source applications enables QXO to support many instances of QAD EA.

One advantage of having a single instance of QXtend support multiple QAD EA instances is that all messages/data are flowing through a single point; this makes monitoring and tracking the data flowing through the organization much simpler as it is all tracked in a central location.

Business Objects



The business object is a way to get data from any QAD application into QXtend. A business object defines the data components required for a complete view of a data entity in any QAD Enterprise Application.

For example, a sales order has a header, some lines, comments and tax information, plus other data components. A sales order business object must encompass all these data components. However, it is not enough to simply list the components—we also must define the relationship between components.

For example, an order can have zero or more lines; the link between order and line is by domain and order number. A complete business object defines all data components, the relationships between the data components, and the fields/attributes of each data component.

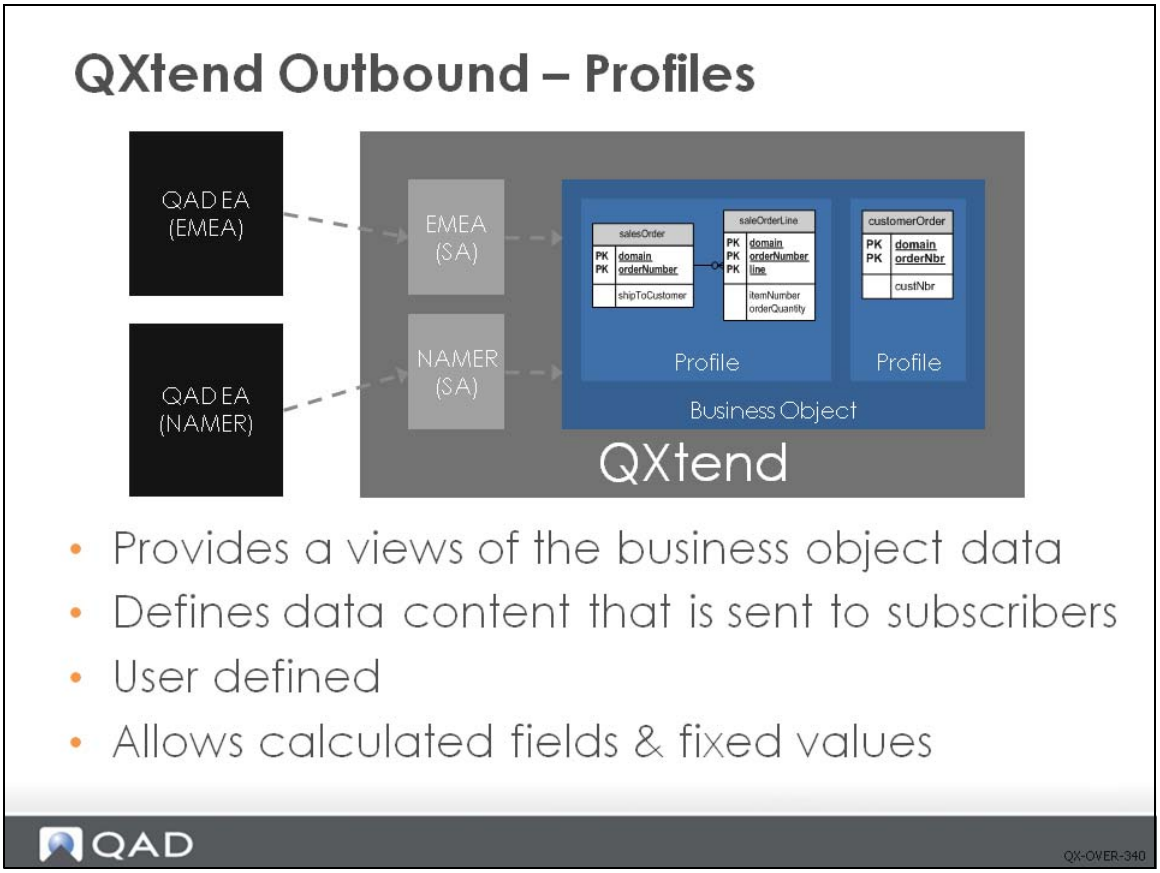
QXO has two types of business object:

- User-defined: Allows the user to build a business object from tables and fields in any QAD Enterprise Application that has a Progress database. The user selects the tables that are part of the object and defines the joins between each of the tables that have been added. The user then selects the fields from each of the tables that must be part of the business object. The definition of the business object drives the extraction of data from a source application instance.
- Application-defined: Allows a QAD application to define exactly what data is published to QXtend. This is used primarily for non-Progress based applications, Enterprise Financials, or complex business objects that cannot easily be mapped directly to the database. Application-defined business objects are imported into QXO via a schema definition that describes the data

components, fields, and relationships. This type of business object cannot be modified by a QXO user. The other major difference is that QXtend does not extract the data for these types of business objects; instead the QAD application publishes the data to QXO.

QXO stores raw data that matches the business object definition for each instance of it in the source application that has been processed by QXO. This data controls the other QXO processes that publish and send data to subscribers. When using user-defined business objects you should ensure that the definition only contains data that is required to minimize the amount of instance data stored in QXO.

Profiles



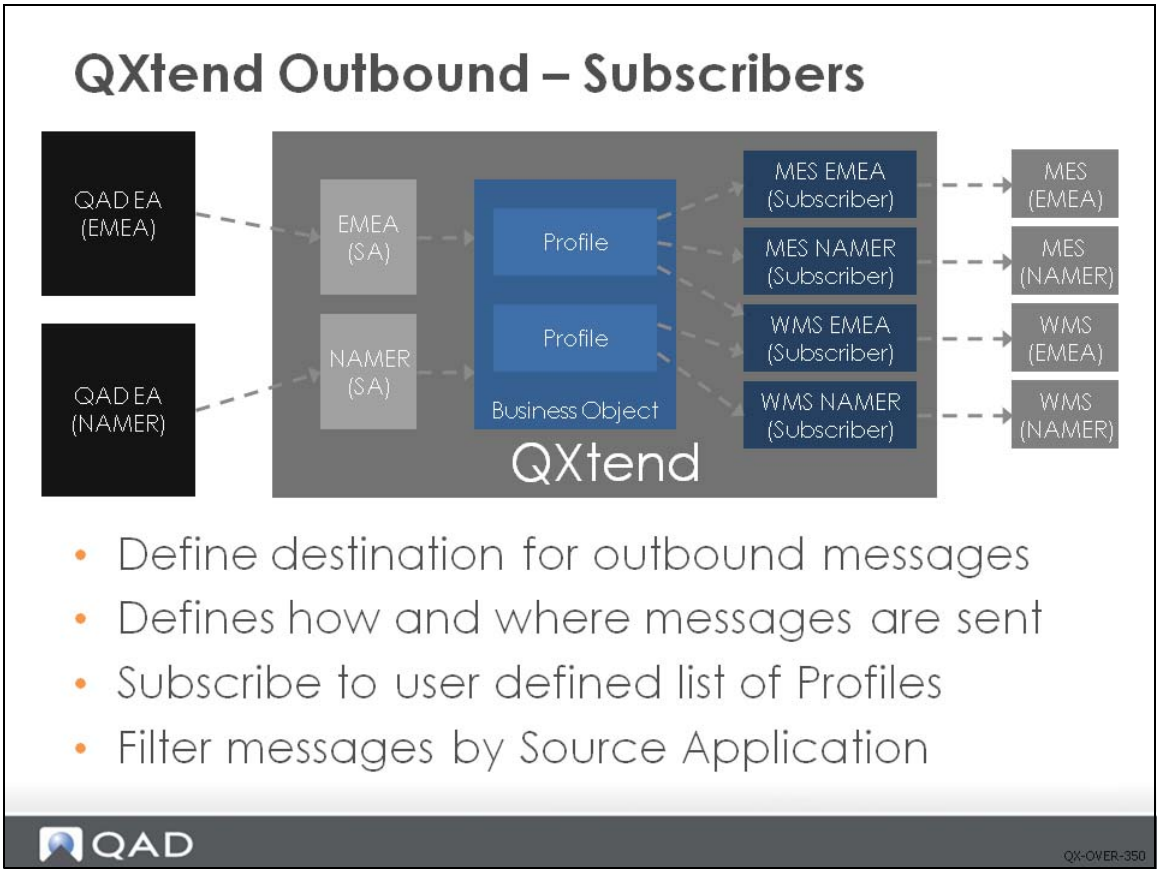
The business object contains the base set of data required by the subscribers. However, not all subscribers need all of the data in the business object: the structure of the data may need to be different for each interested subscriber. QXO uses *profiles* to provide data to the subscriber that meets the subscribers' data, format, and structure requirements.

Profiles allow the user to map data from the business object into a different view. Typically profiles contain a subset of the business data object data that can be enriched with calculated fields and fixed values. The user can change the name of the XML nodes published to the profile, as well as take child tables and publish them as part of the parent table. The mapping capabilities provided by the profiles cover most scenarios; however, it is not a complete mapping tool and some scenarios cannot be mapped by a profile. In these scenarios, mapping is performed by a transformation tool as a separate step.

QXO allows many profiles to be created against a business object; this enables a single business object to publish data in the different formats required by subscribing applications. The QXO interface allows users to create and modify profiles against any business object without the need for coding. Coding is only mandated if complex business logic is required to calculate the value of any field in the profile; these fields are called *calculated fields* and execute Progress code to calculate the field value.

To summarize, profiles map business object data into the format required by the application that is going to subscribe to the data.

Subscribers



QXO provides a configuration toolset to facilitate the delivery of QAD EA data to other QAD or non-QAD applications. A *subscriber* in QXO is a named destination point for messages: it identifies the target (destination) application in QXO, rather like the source application identifies the data source. A subscriber is created for each target application instance that wants to receive data from QXO.

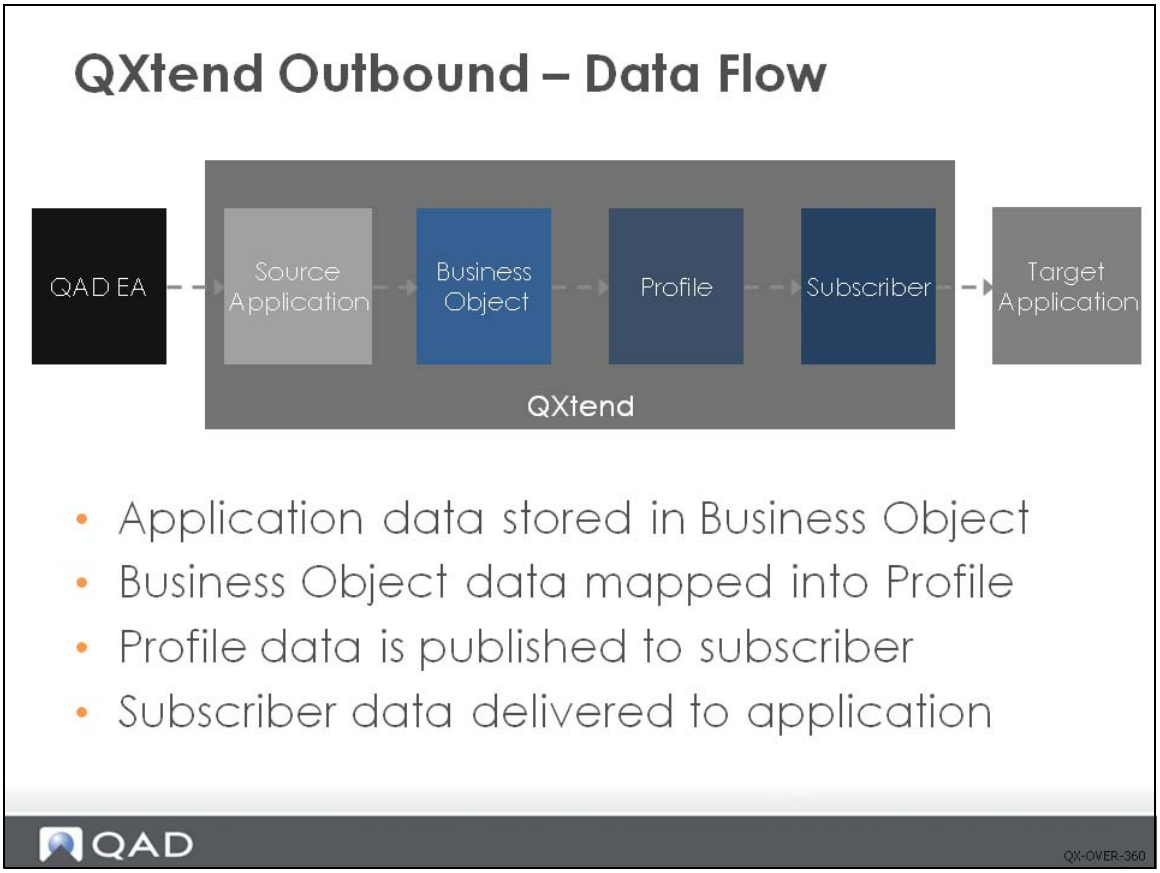
The subscriber configuration describes how and where the data is sent. QXO supports two delivery options:

- File Drop: Allows the user to specify a directory into which the messages that are subscribed to are delivered. The directory must exist on the server where the QXO processes are running.
- Web Service and QXI Web Service: Allows data from QXtend to be used to invoke a Web Service call by passing SOAP XML over HTTP to the target application. The easiest Web Service destination to configure is QXI; however, you can send messages to other Web Services or HTTP listeners.

The subscriber setup also defines the list of profiles to be published to that particular subscriber. In this way the subscriber can select the data that it is interested in and the format of the data it receives. A subscriber must subscribe to a least one profile, but it can subscribe to many if required.

The subscriber can select the source applications it receives data from to ensure it only gets data from the instance it is integrating with. Subscribers also can restrict the data subscribed to by source domain and source entity. This ability provides a highly configurable subscription mechanism for application integration.

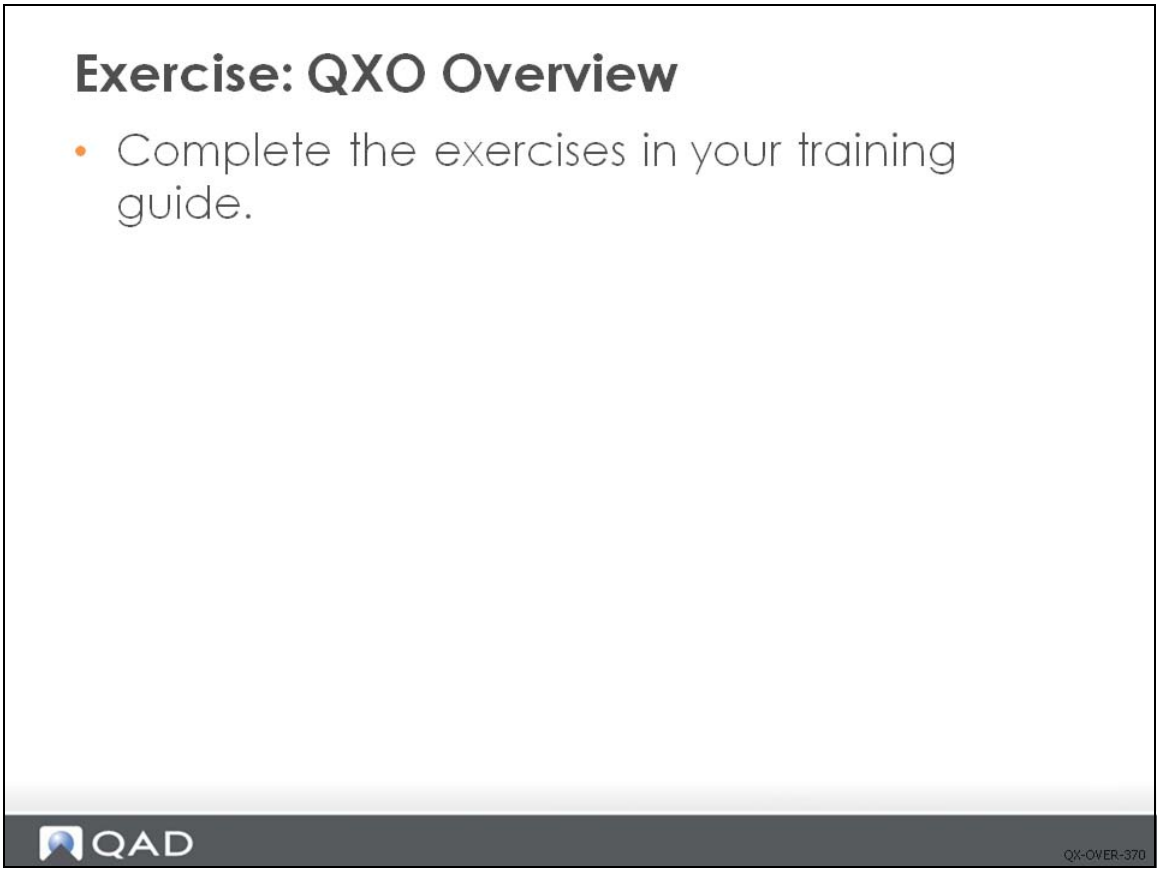
Data Flow



The steps to move data from a QAD Enterprise Application to a subscribing application are as follows:

- Extract: Changes to data in a QAD Enterprise Application are extracted or published for a source application, the data matches the structure defined in the QXO business object definition, and the extracted data is stored in QXO as an instance of the business object.
- Publish: The stored business object instance data is mapped into the profiles that have been defined against the business object. If there are any calculated fields or fixed values defined against the profile, these are calculated/assigned as part of this step.
- Deliver: The profile messages generated in the previous step are passed to each subscriber that has registered an interest in the profile message. The message is delivered to the target application using the details configured on the subscriber.

Exercise: QXO Concepts



The following list shows a number of key terms and concepts used in QXO. In each statement below, fill in the correct term from the list. Some terms may be used more than once

subscriber	business object
source application	extract
publish	Direct Data Publish (DDP)
interested subscribers	XML QDocs
Profiles	Deliver

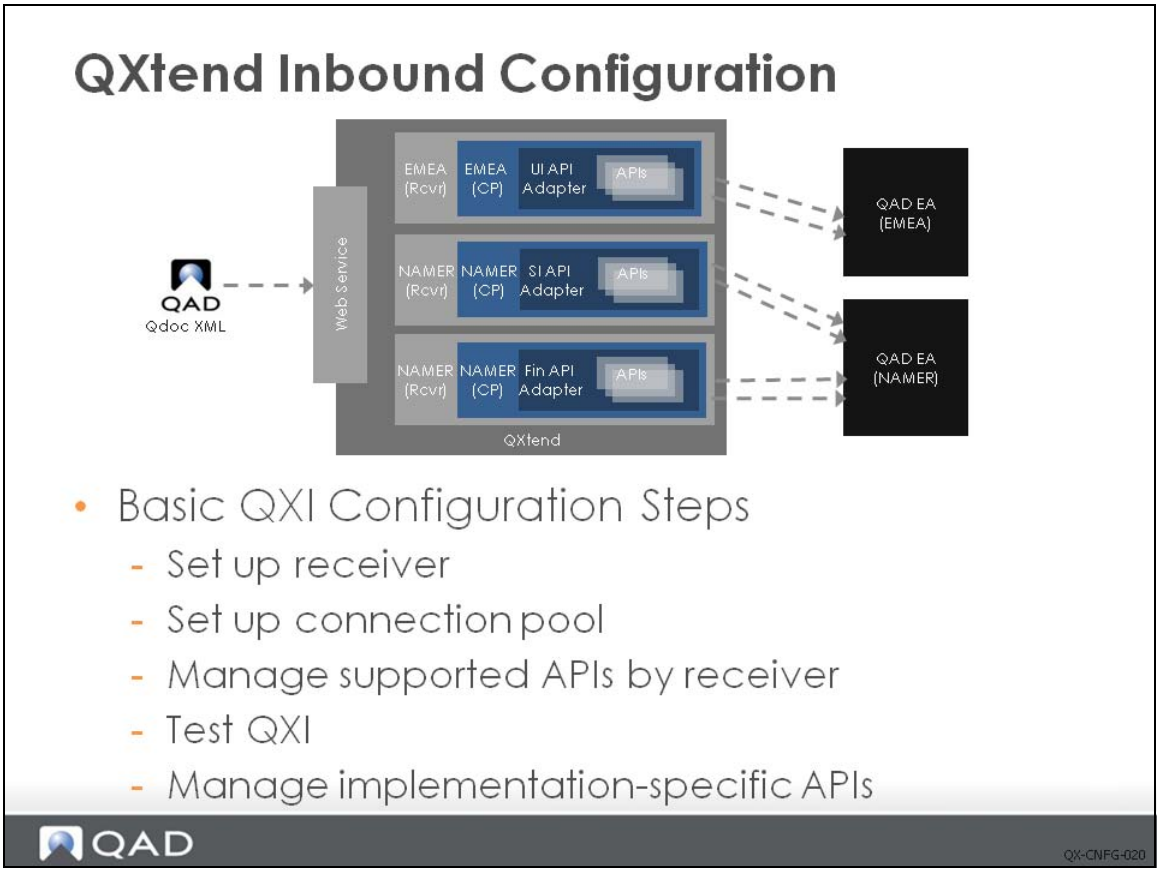
1. QXtend Outbound (QXO) enables QAD Enterprise Applications to publish application data to _____.
2. Data is published as _____.
3. Data from the _____ is first moved to a _____, which is a subset of the data stored in the source application.
4. QXO uses _____ to provide data to the subscriber that meets the subscribers' data, format, and structure requirements.
5. A _____ in QXO is a named destination point for messages.
6. The steps to move data from a QAD Enterprise Application to a subscribing application are 1) _____; 2) _____; and 3) _____.

7 For _____, data is not extracted from the source application by QXO.

Chapter 2

QXI Configuration

Overview

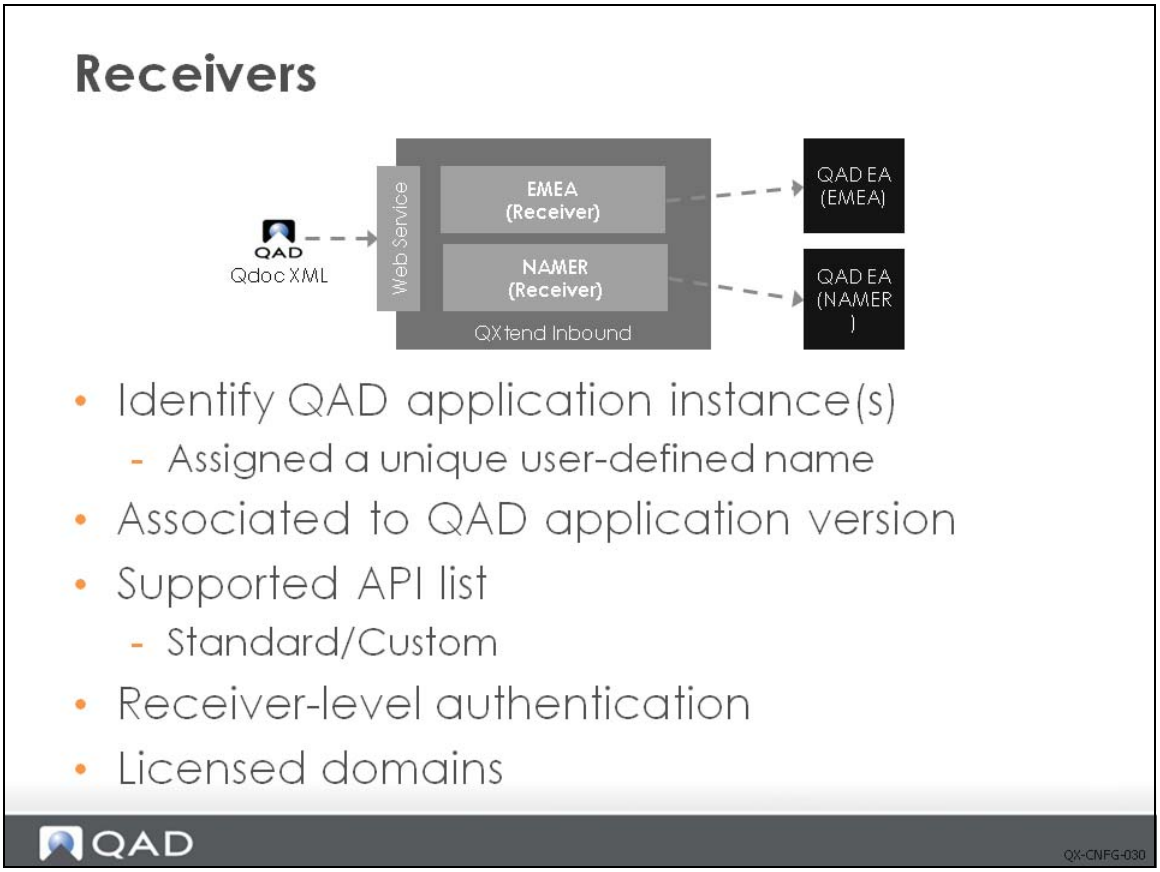


The QXtend Overview slides described the basic components and concepts behind QXI, but did not describe QXI setup and configuration. This training section describes in detail the QXI components, and their configuration and management.

To ensure that messages can be processed into a target QAD application, you must configure QXI by following several basic steps; for example, you must create the services and components that drive QXtend. To configure QXI, you must do the following:

1. Set up the receiver. This step identifies the instance of the target QAD application (application type, version, and supported APIs).
2. Set up a connection pool. This step specifies the connection details required to connect and process requests into the target application.
3. Manage the supported receiver APIs. This step includes add, modify, and delete support for APIs used by the receivers configured in QXI.
4. Verify your configuration. This step validates the QXI configuration and ensures that a request can be processed into the target application.
5. Manage implementation APIs. This step manages the APIs required during the implementation project. These APIs are typically customer-specific.

Receivers



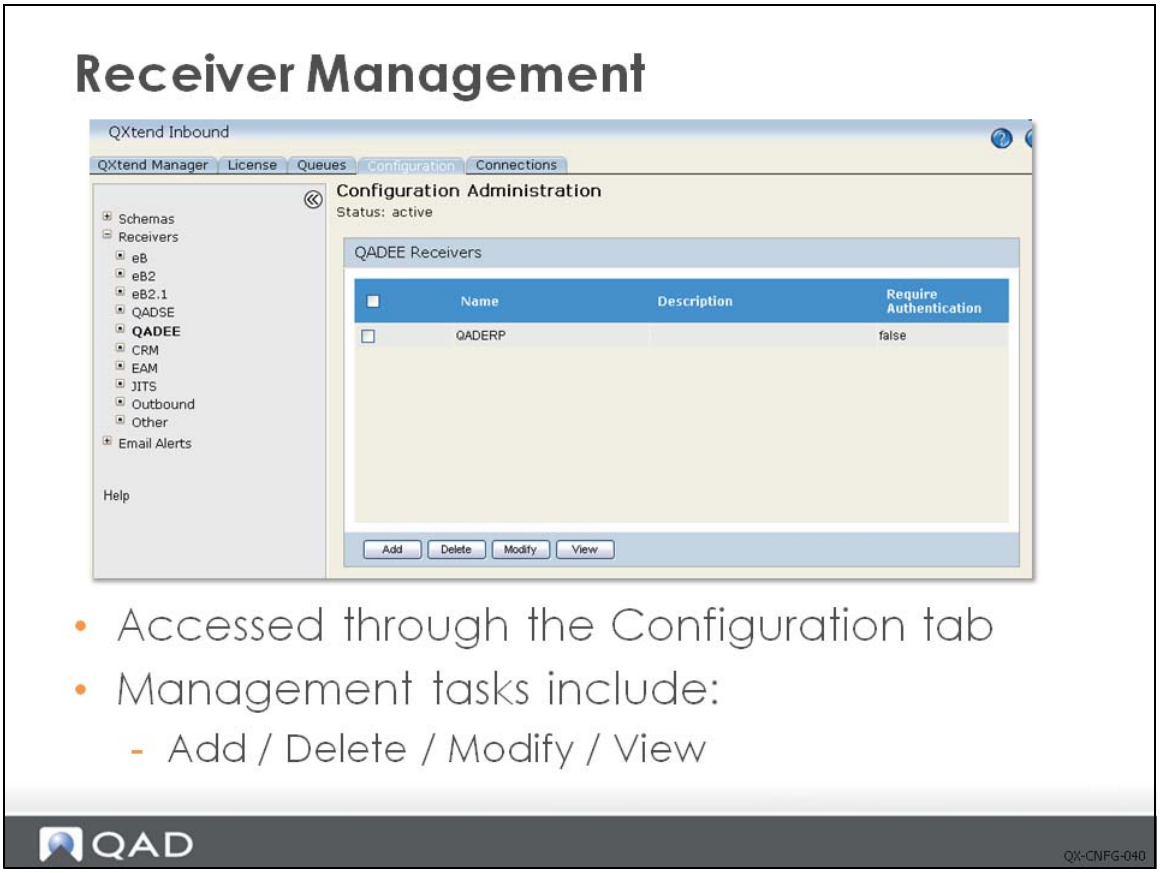
The concept of a receiver is core to understanding QXI. A receiver serves four primary purposes:

- Identifies the application instance. QXtend will support many instances of QAD applications; therefore a QXtend installation is not required for each and every QAD application instance that is using QXtend. However, if QXtend is to support many QAD application instances, there must be a way to differentiate between those instances. The receiver allows each QAD application to be assigned a name that uniquely identifies it. For a request to be successfully processed by QXI, a valid receiver must be included in the request.
- Identifies the QAD application version. An important attribute of the receiver is the application version, which identifies the exact release of the application to which the receiver is linked. Identifying the release is important as this identifies all of the APIs that are available through QXtend, as well as the versions of those APIs that are available for that specific application release.
- Defines the APIs supported. Each receiver must define the APIs that are valid for use. QXtend ships with hundreds of APIs available for use in any integration project, and typically only a subset are used for an integration. QXtend allows the user to define which APIs are available to be used—or, in QXtend terms, supported—by that receiver. This helps to prevent unauthorized use of the API and enables how QXtend is being used to be tracked.
- Receiver-level authentication. This attribute is used to indicate that this receiver requires all QDocs to contain authentication information. For each receiver in the system, you can specify that every QDoc for the receiver must contain authentication credentials consisting of a username and password, or a session ID.

- Determines licenses and licensed domains. The QXtend pricing model requires all QAD applications that receive data from an external application to be identified. These applications count as integration points and are part of the overall price that is charged for QXtend.

  When a receiver is added to QXI that is for the eB or eB2 version, the Licensed check box must be selected if it will receive data from an external applications. When a receiver is added to QXI that is for the eB2.1 (or later) version, a list of the domains in that instance that will receive data must be provided. This information is then used when processing a request from an external system to ensure that the target is licensed to receive data. If the target is not licensed to receive data, the message is rejected.

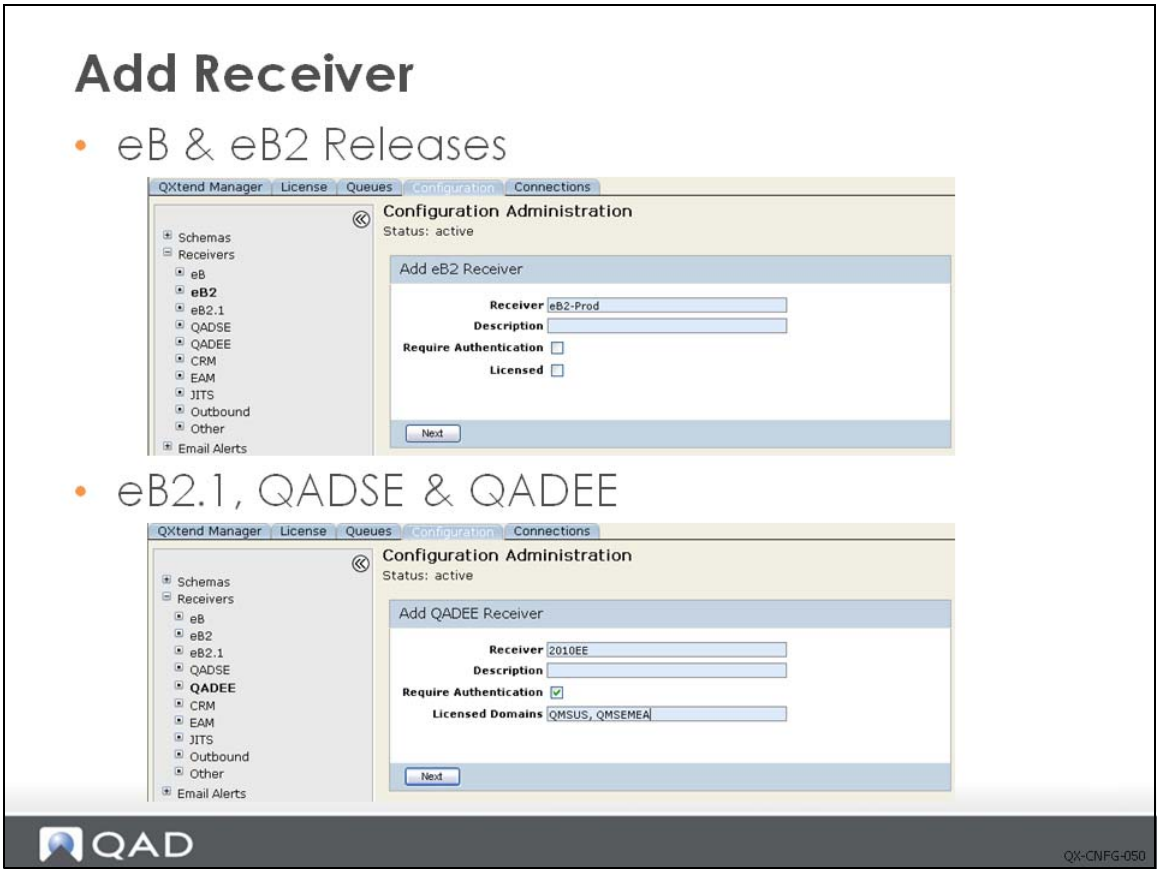
Receiver Management



Receivers are managed through QXI. All receiver management tasks are accessed on the Configuration tab via the Receivers node on the tree menu.

Receivers are grouped by application version. To perform tasks against a specific receiver, select the application version from the tree menu, select the receiver, and then click a button—Add, Delete, Modify, View and so on—to choose the action to perform against the receiver.

Add Receiver



Receivers are created against specific application versions selected from the Receiver menu. Select the version, then click Add to add a new receiver. When adding a new receiver, a unique receiver name must be specified; this is the name used to route requests to the correct QAD application instance.

Set Require Authentication to True if it is required that every Qdoc for this receiver must contain authentication credentials consisting of a username and password, or a session ID.

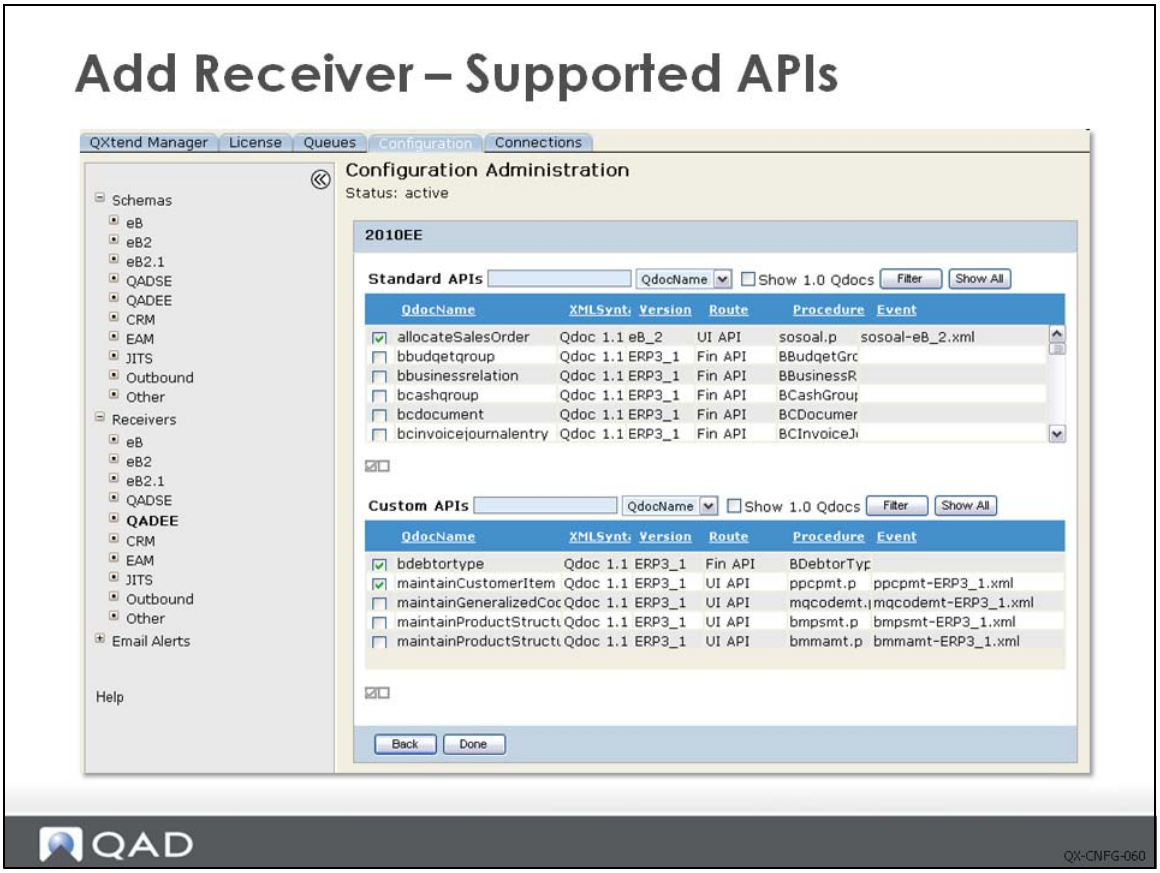
This attribute is used to indicate that this receiver requires all QDocs to contain authentication information. For each receiver in the system, you can specify that every QDoc for the receiver must contain

If the receiver will receive requests from external non-QAD applications, this must be specified. For eB and eB2 releases, select the Licensed check box to indicate that requests from external applications can be processed. For all other application versions, a list of the domains that will receive data from external applications must be provided. If the licensed domains information is not provided, QXI rejects messages from external applications to the receiver/domain.

You must specify name, application version, authentication and standard and/or custom QDocs that are supported. You may also have to specify licensed domains if the target application supports domains.

You choose the QDocs that are supported from a table that contains all QDocs that are available for the specific application version of the receiver.

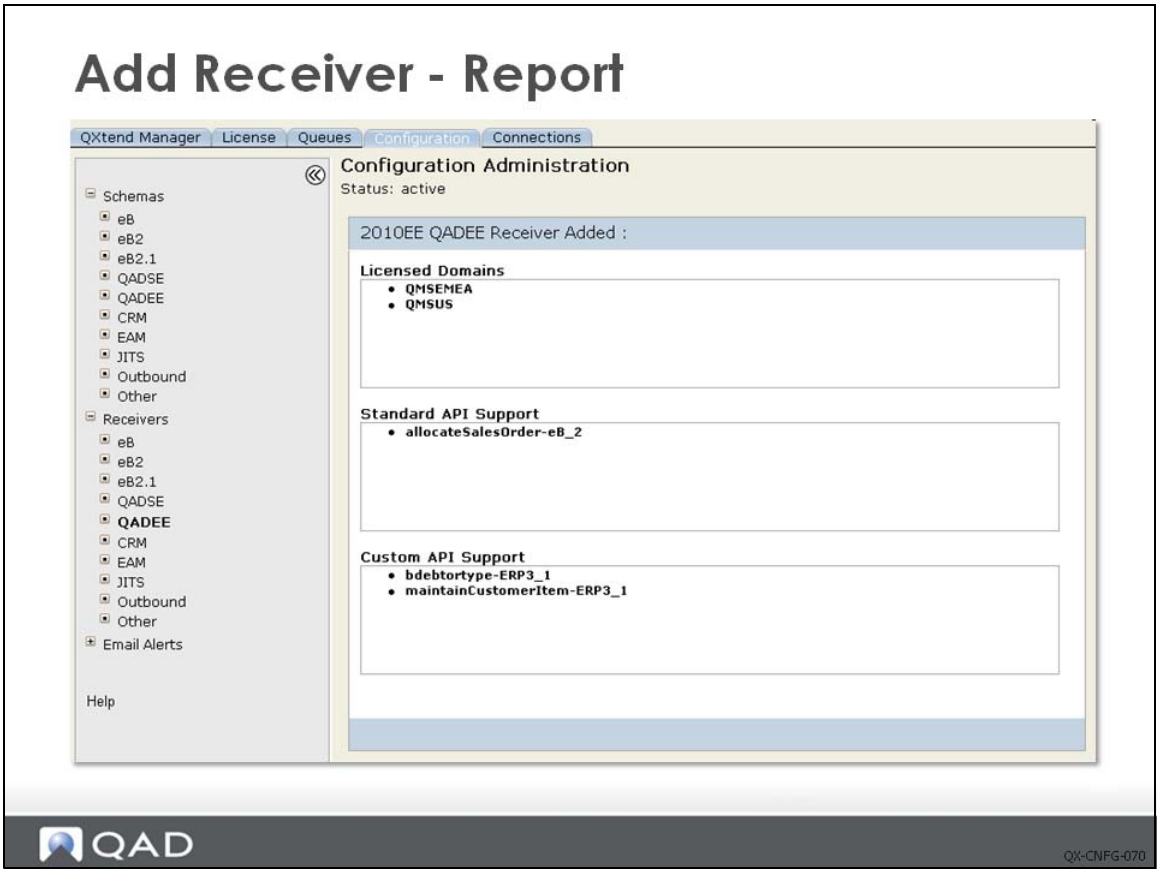
Supported APIs



When a new receiver is created, you need to select the APIs that will be supported by the receiver. The list of supported APIs enables QXtend to limit the API transactions that can be processed by the receiver. This ability is useful when you need to limit the use of QXtend to approved transactions only. If all APIs are to be supported, select the Select All check box to add all of the APIs to the receiver's supported API list.

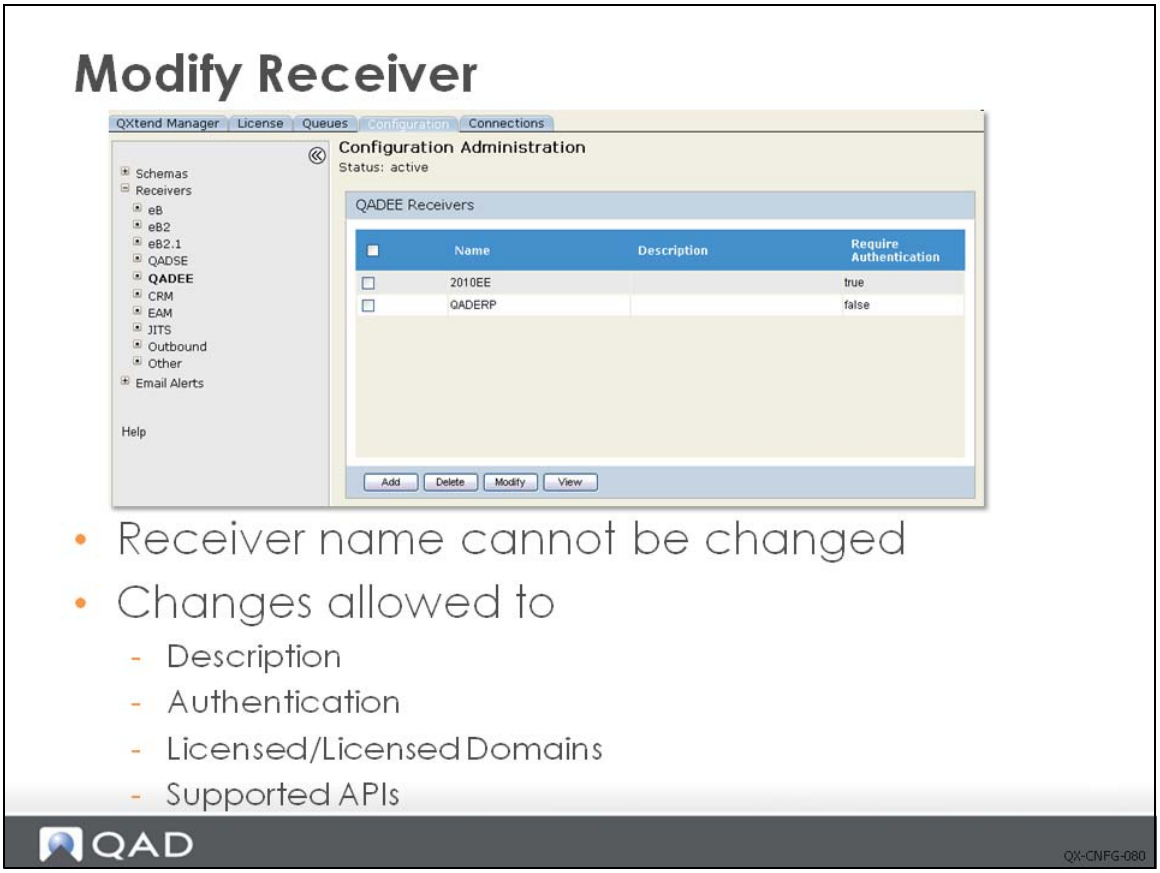
The system displays two lists of APIs. The Standard APIs list contains all APIs provided by QAD for use with QXtend. The Custom APIs list contains the APIs developed specifically for the customer. Select the APIs you want to support, then click Done.

Receiver Report



After successfully adding a receiver, a summary report displays showing licensed domains, standard API support, and custom API support.

Modifying a Receiver



You cannot change the name assigned to a receiver. If you need to change the name of a receiver, you must delete the receiver and then re-create it.

You can change the description of a receiver and Require Authentication setting for a receiver.

You can change the License settings for a receiver; depending on QAD application release this can be done using a simple check box or maintaining the list of licensed domains. To modify the settings on a receiver, select the check box next to the name of the receiver and click Modify Licensed.

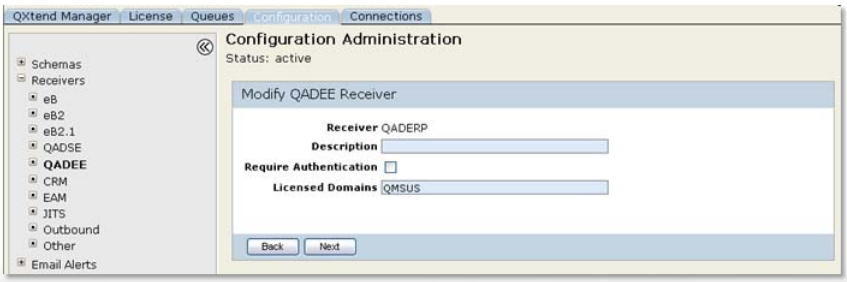
The list of supported APIs can also be modified—select the check box for the Receiver and click Modify API.

Modify Receiver

- eB & eB2 Releases



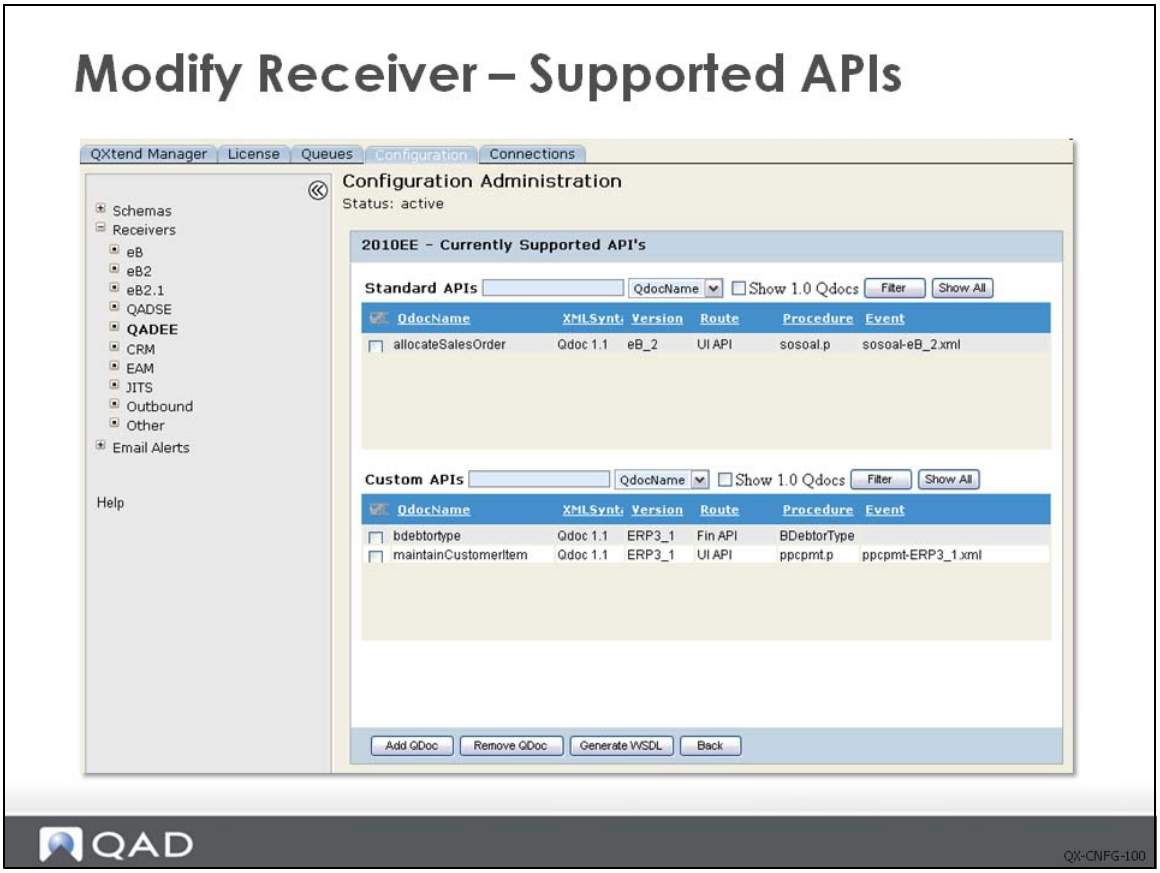
- eB2.1, QADSE, QADEE



QAD

QX-CNFG-090

Modifying a Receiver - Supported APIs

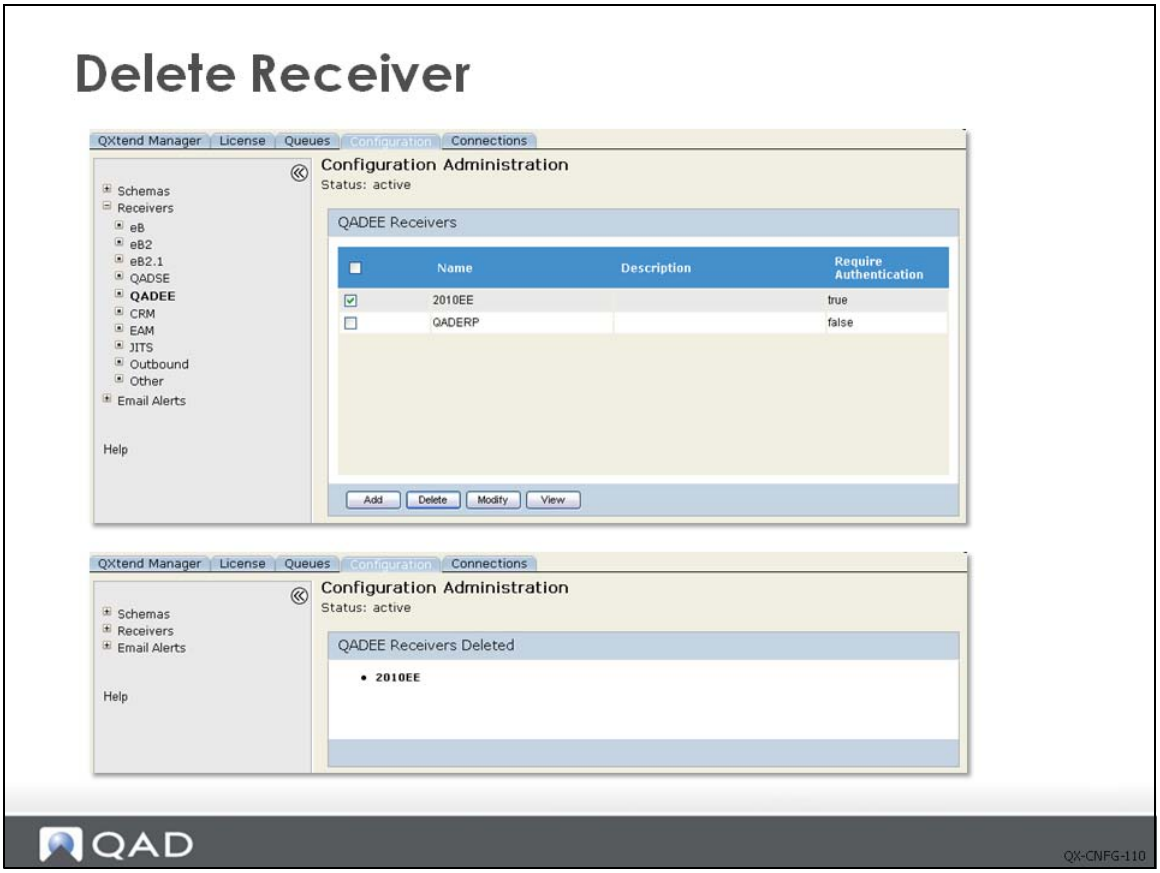


You also can modify the list of APIs supported by the receiver: you can add support for additional APIs, or remove support for APIs that are no longer required.

To add support, click Add QDoc. A new screen displays that lists the APIs that are currently not supported by the receiver. Select the APIs to add and click Finish.

To remove support, select the check box next to the APIs that need to be removed and click Remove QDoc.

Deleting a Receiver

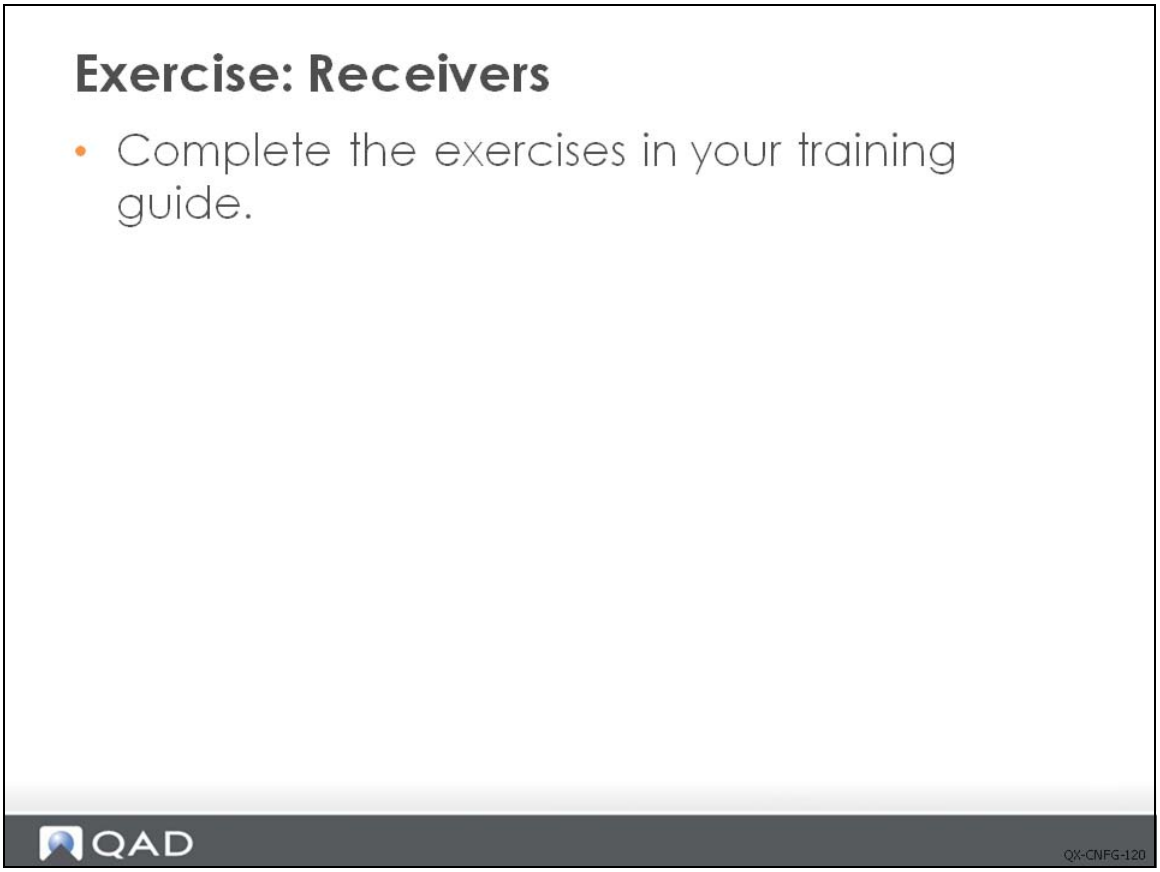


You can delete receivers as required. Deleting a receiver removes all traces of the receiver from the QXtend system.

To delete a receiver:

1. In the Configuration Manager, choose Receivers|<*mfgpro-version*>, where <*mfgpro-version*> is eB, eB2, eB2.1, QADSE, or QADEE. Other products may display on this menu. 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, "Please close all files within specified receiver(s)?" Choose OK if no documents are in transit for the receiver. If QXI is active, the suspend options display. Choose a suspend option to cancel or continue.
5. If you suspended QXI services in step 4, you have the option of resuming QXtend Manager at this point. If you do this, QXI returns to the active state and a confirmation displays.
6. To exit the delete receiver confirmation screen or the resume QXI services confirmation, choose any other menu option in the QXtend Manager interface.

Exercise: Receivers

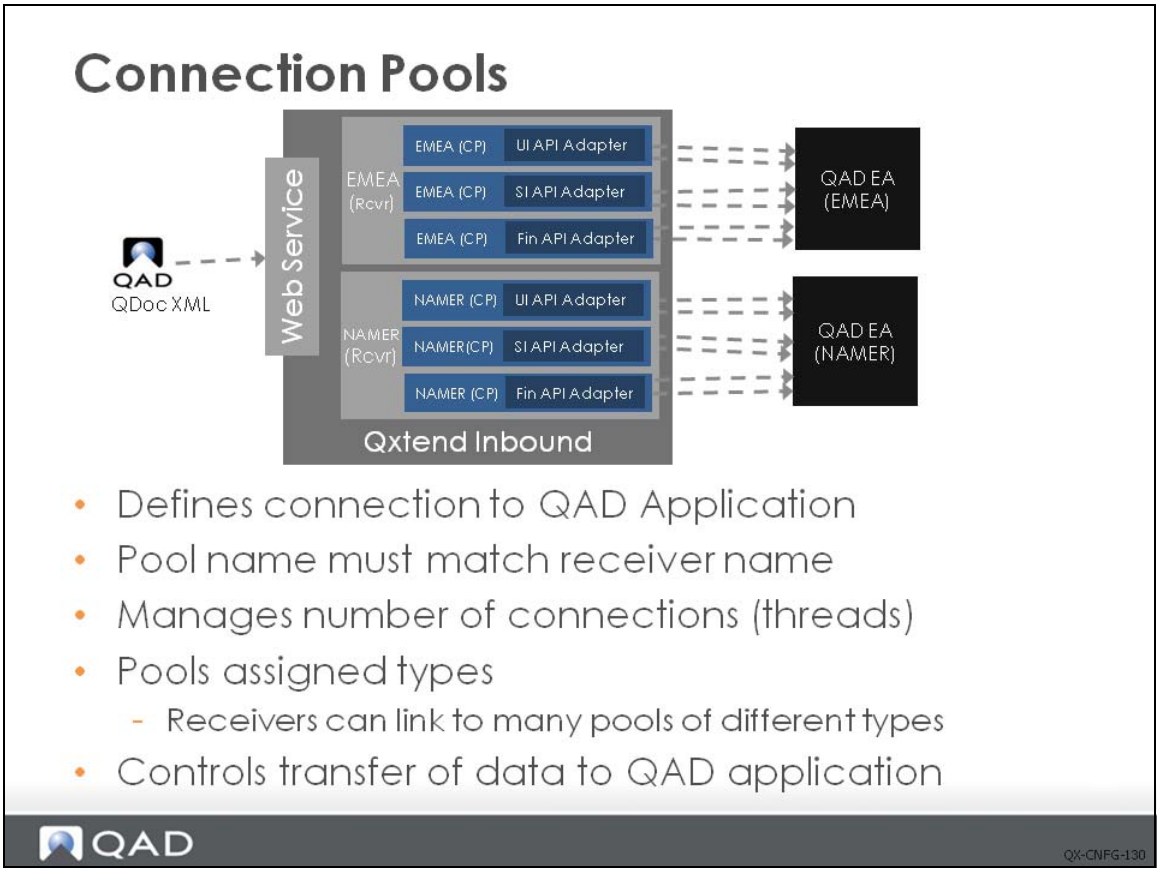


The following list shows a number of key concepts used in receivers in QXI. In each statement below, fill in the correct term from the list.

Configuration tab	authentication
license settings	Receivers node
application instance	supported APIs

1. A receiver serves five primary purposes: Identifies the _____, identifies the QAD application version, defines the authentication, defines the APIs supported, and determines the licenses and licensed domains.
2. All receiver management tasks are accessed on the _____ via the _____ on the tree menu.
3. The list of _____ enables QXtend to limit the API transactions that can be processed by the receiver.
4. You can change the description, _____ and _____ for a receiver.

Connection Pools



QXI connection pools provide connections from QXtend to specific instances of QAD applications. The connections enable request data passed into QXtend to be loaded into the target QAD application using the APIs that are available. The collection of connection pools available in QXI is managed by the Connection Pool Manager.

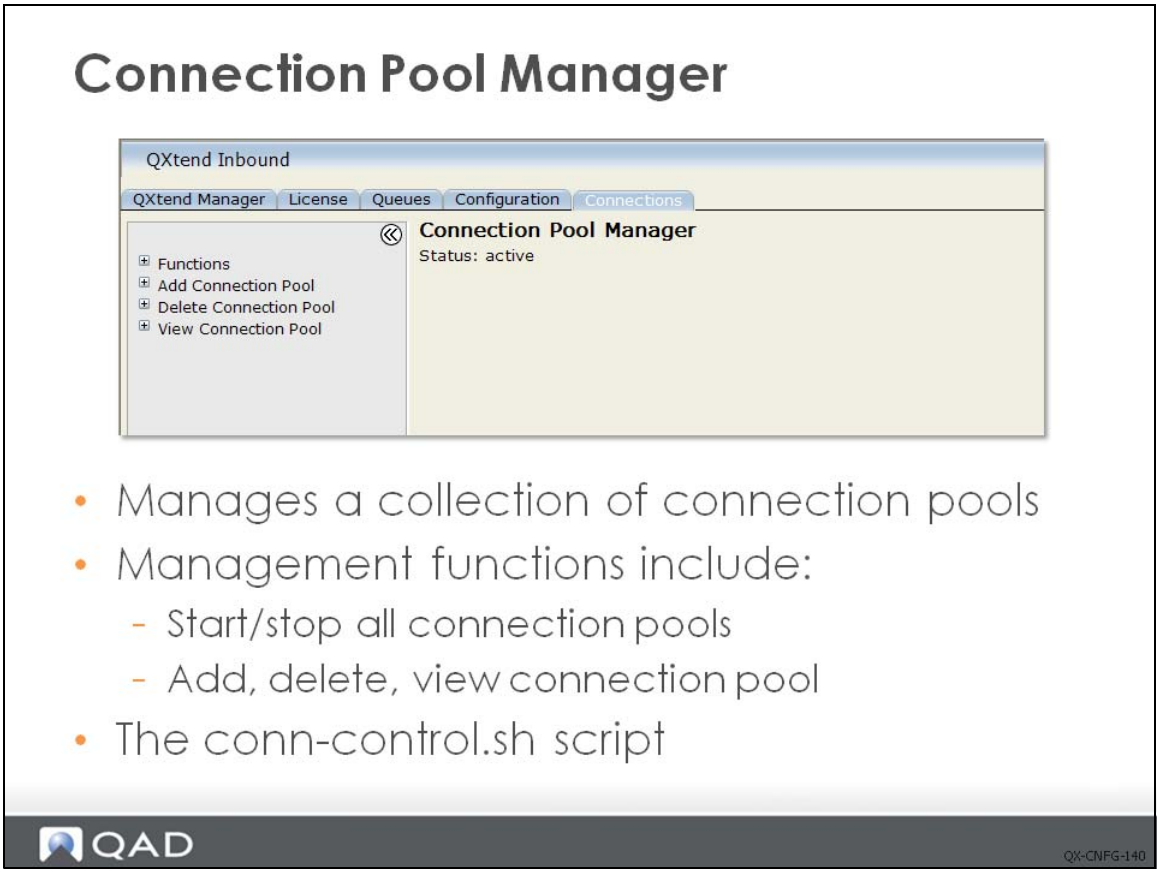
Connection pools are assigned a name that must match the name of the receiver it is intended to process data for. If the names do not match, QXtend cannot pass data from the receiver to the connection pool that ultimately loads the data into the target application.

The settings on the connection pool also control the number of connections that are available to process requests; the minimum and maximum pool size are standard settings on a connection pool. Once the maximum number of connections has been reached, no new connections are created. The maximum pool size settings shows the maximum number of requests that can be processed simultaneously.

There are different types of connection pool; the connection pool type controls how data is moved from QXtend to the target application; for example, the UI API uses a telnet connection to load data, whereas the SI API uses a connection to the Progress AppServer and the Fin API is one kind of SI API but use a connection to Financial AppServer. When creating an instance of a connection pool, the details required to establish the connection to the QAD Application can vary considerably depending on the connection pool type.

The combination of the connection pool name and pool type uniquely identifies a connection pool. A receiver can be linked to one or more connection pools depending on the APIs used. This allows requests to one receiver to use different API mechanisms to access the QAD application logic. The relevant connection pool required to process the request is identified from the API name, and version details that are stored within QXtend.

Connection Pool Manager

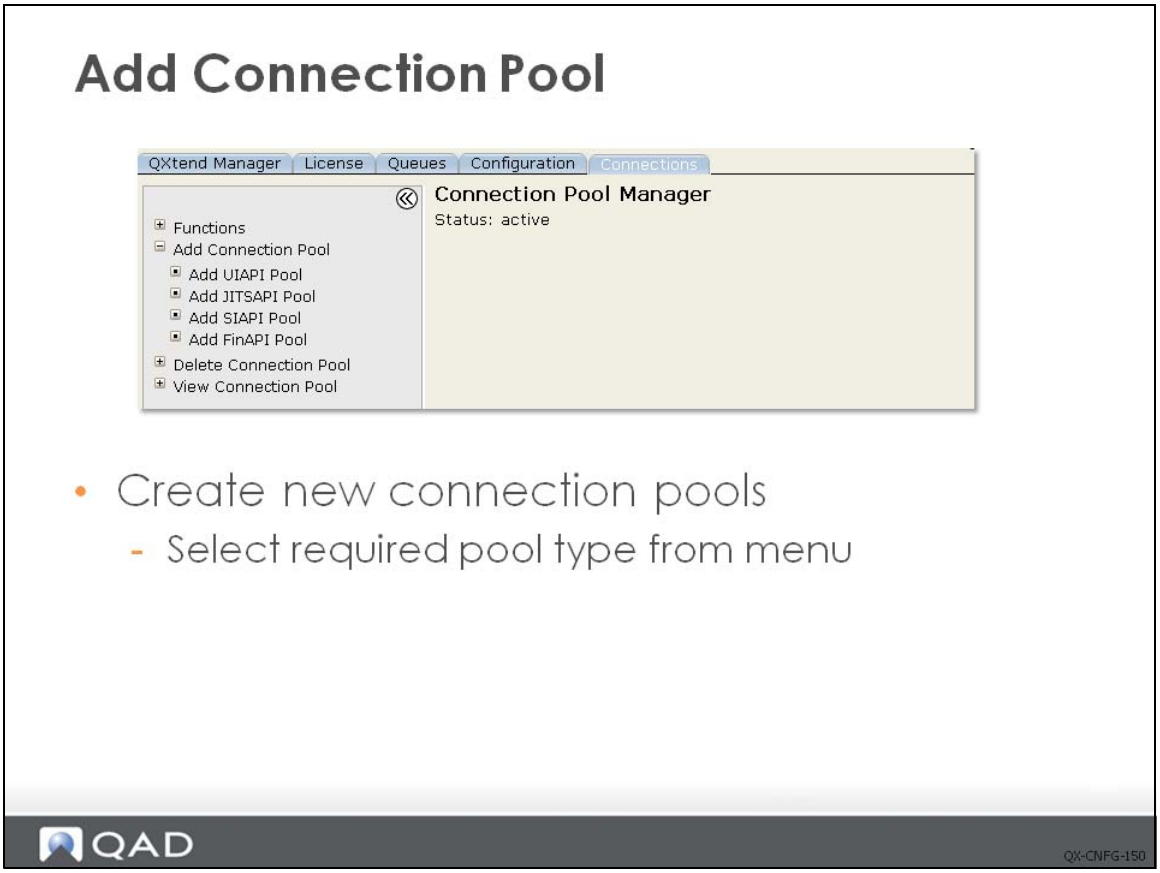


You access the Connection Pool Manager by selecting the Connections tab. The Connection Pool Manager provides a centralized management console that allows users to maintain their required connection pools. The target QAD applications may require several connection pools per instance depending on the APIs being used.

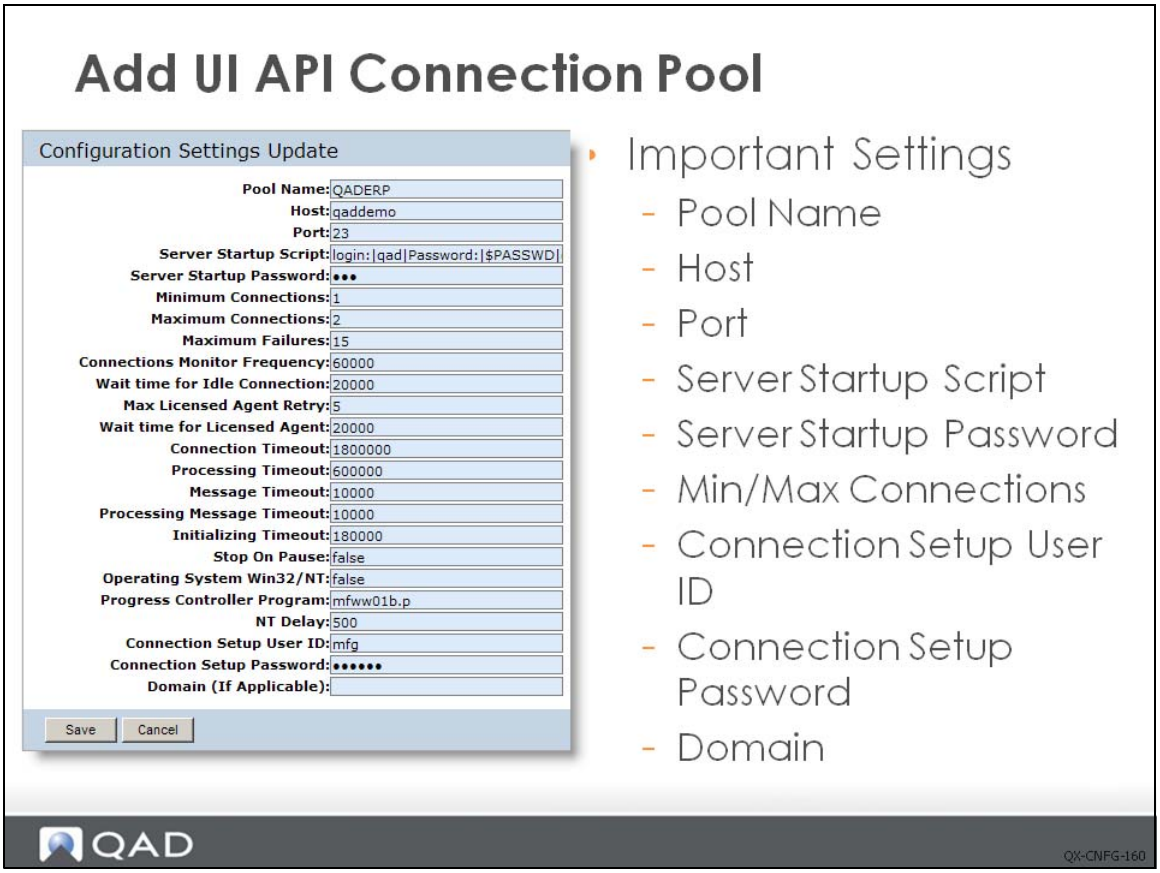
Connection pools can be individually created, deleted, updated, stopped, or started from this console. You also can stop and start all connection pools by clicking Restart Connection Pool Manager.

In addition to managing connection pools through Connection Pool Manager in the administrative UI, you can also use the connection pool manager script to start, stop, restart, and shut down connection pools, as well as query the connection pool status. This script can be found under `WEB-INF/scripts` directory of QXI web application.

Adding a Connection Pool



Adding UI API Connection Pool



The UI API connection pool type leverages a telnet connection to log in to the QAD application and load data from QXtend. Many configuration parameters are defined against the UI API connection pool; however, most of these settings do not need to be changed as they are for advanced performance tuning and configuration (for details see *Technical Reference: QAD QXtend*).

The following parameters must be configured correctly to establish a connection:

Pool Name. – The name that you want to assign to the connection pool instance. Remember that this must be exactly the same name as the Receiver and it is also important to remember that QXtend Inbound is Case Sensitive so the names must match exactly including the case of the letters.

Host. Hostname of the machine where the QAD Application instance is running, the UI API Adapter opens a telnet connection to the host machine and use the name given here to establish that connection.

Port. The port number that is available on the host machine for telnet communications. The default is 23 which is the typical default for telnet communication.

Server Startup Script. – When UI API Adapter opens a connection, it uses this script to login to the host machine and launch a QXtend specific startup script. This setting scripts the actions that must be performed to successfully login and start the connection. The script is effectively prompt value pairs the first entry is a prompt that is displayed during each step of the login process and the second entry is the value that needs to be entered. The prompts are simple to

identify by using a telnet session to log into the machine and launch the connection startup script, record these prompts and actions and these are the values that need to be entered into this parameter.

The following pre-formatted script is provided that needs to have the correct values substituted when the Add UI API connection pool option is selected:

```
loginPrompt|userid|passwordPrompt|$PASSWD|osPrompt|startScript
```

For a valid connection pool login script, it should be changed to something like this:

```
login:|mfg|Password:|$PASSWD|$|/dr01/scripts/client.qxtend
```

Note The password is not supplied as plain text to ensure that the password is protected.

Server Startup Password. – holds the password that is used by the Server Startup Script, the value of the password is masked.

Minimum Connections. – the minimum number of idle connections that the connection pool will try to maintain. If the minimum connections is 2 and the pool has 2 idle connections, and then one of the connections is assigned to the busy status the connection pool will automatically start another connection to maintain a minimum of 2 idle connections.

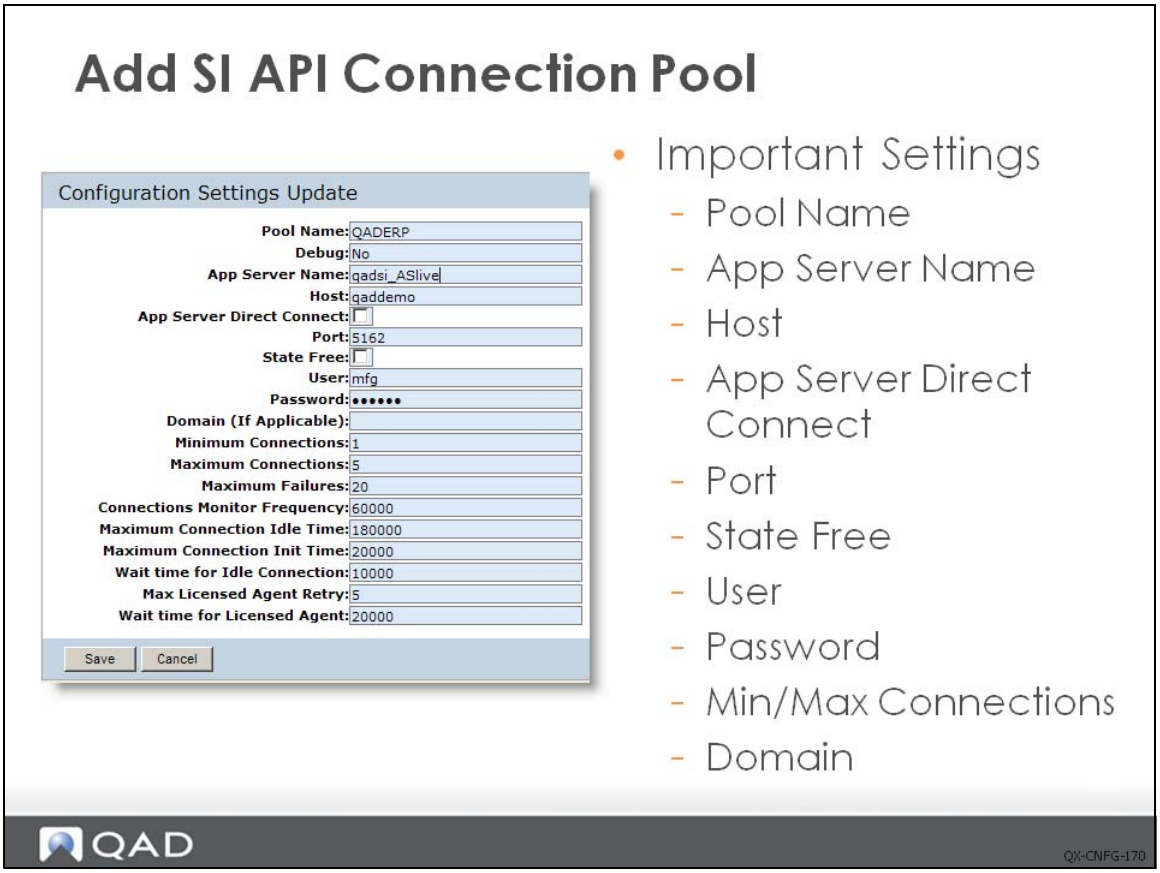
Maximum Connections. – the maximum number of connections that can be created for this connection pool. Once the maximum number of connections has been reached the connection pool stops trying to spawn new idle connections.

Connection Setup User ID. – the user id that is used to login to the QAD Application. The startup script defines the user to connect to the host machine and this is the QAD Application user.

Connection Setup Password. – Password required for the specified user id to login successfully into the QAD Application, this password is not shown in plain test.

Domain. – the QAD Application domain to use when logging into an application version that supports the QAD Domain architecture.

Adding SI API Connection Pool



The SI API connection pool type uses a the Progress AppServer technology to execute QAD Business Services without using a telnet session and therefore requires a different set of connection parameters to access the services. There are many configuration parameters that are defined against the SI API connection pool. However, many of these settings do not need to be changed as they are for advanced performance tuning and configuration (for more information on the parameters see the *QXtend Technical Reference Guide*).

Pool Name. The name that you want to assign to the connection pool instance, remember that this must be exactly the same name as the Receiver and it is also important to remember that QXtend Inbound is Case Sensitive so the names must match exactly including the case of the letters.

AppServer Name. The name of the AppServer that is hosting the Business Services that need to be accessed by QXtend. This is the name that has been configured in the Progress `ubroker.properties` file.

Host. Hostname of the machine where the Progress AppServer instance is running the SI API Adapter connections to the machine hosting the Progress AppServer and uses the name given here to establish that connection.

App Server Direct Connection. Specify whether you want QXtend Inbound to directly connect to the AppServer.

Port. Enter the port number for the AppServer or the NameServer, depending on your App Server Direct Connect setting. If you selected the App Server Direct Connect option, enter the port number for the AppServer; otherwise, enter the port number for the NameServer that is controlling the AppServer instance.

State Free. Specify in which operating mode the connection pool can connect to AppServer.

Yes: The connection pool can only connect to AppServer when operatingMode is State-free.

No: The connection pool can only connect to AppServer when operatingMode is Stateless.

User. The user id that is to be used to login to the QAD Application instance.

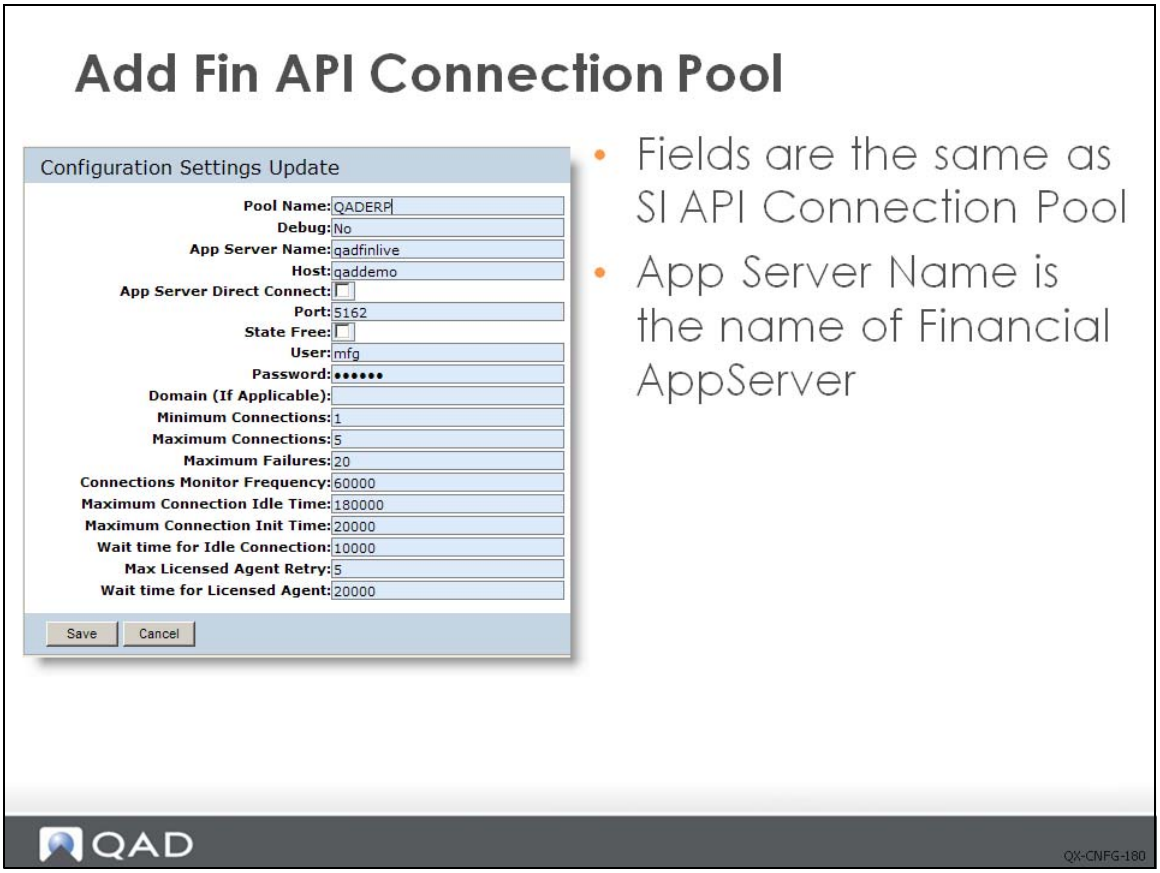
Password. Holds the password that is used to login to the QAD Application, the value of the password is masked.

Minimum Connections. The minimum number of idle connections that the connection pool will try to maintain. If the minimum connections is 2 and the pool has 2 idle connections and one of the connections is assigned to the busy status, the connection pool will automatically start another connection to maintain a minimum of 2 idle connections.

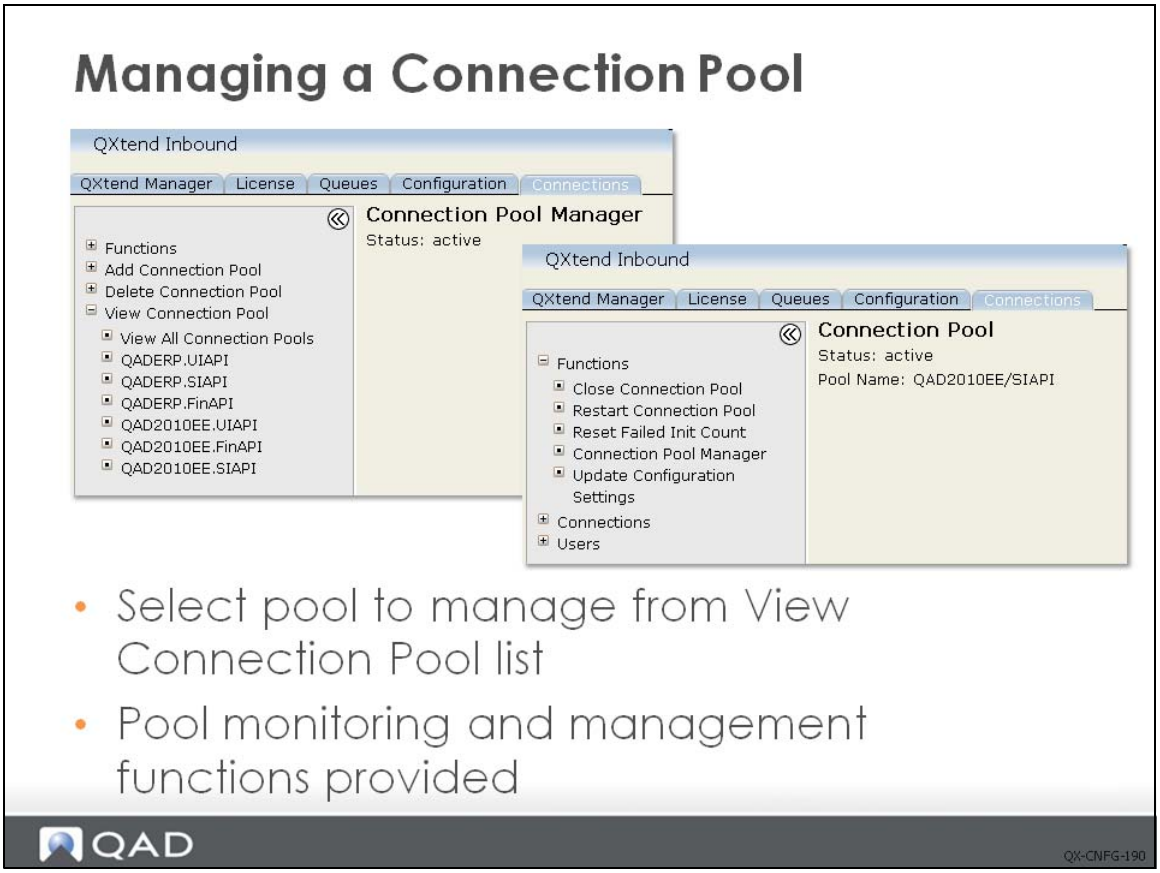
Maximum Connections. The maximum number of connections that can be created for this connection pool. Once the maximum number of connections has been reached the connection pool stops trying to spawn new idle connections.

Domain. The QAD Application domain to use when logging into an application version that supports the QAD Domain architecture.

Adding an Fin API Connection Pool

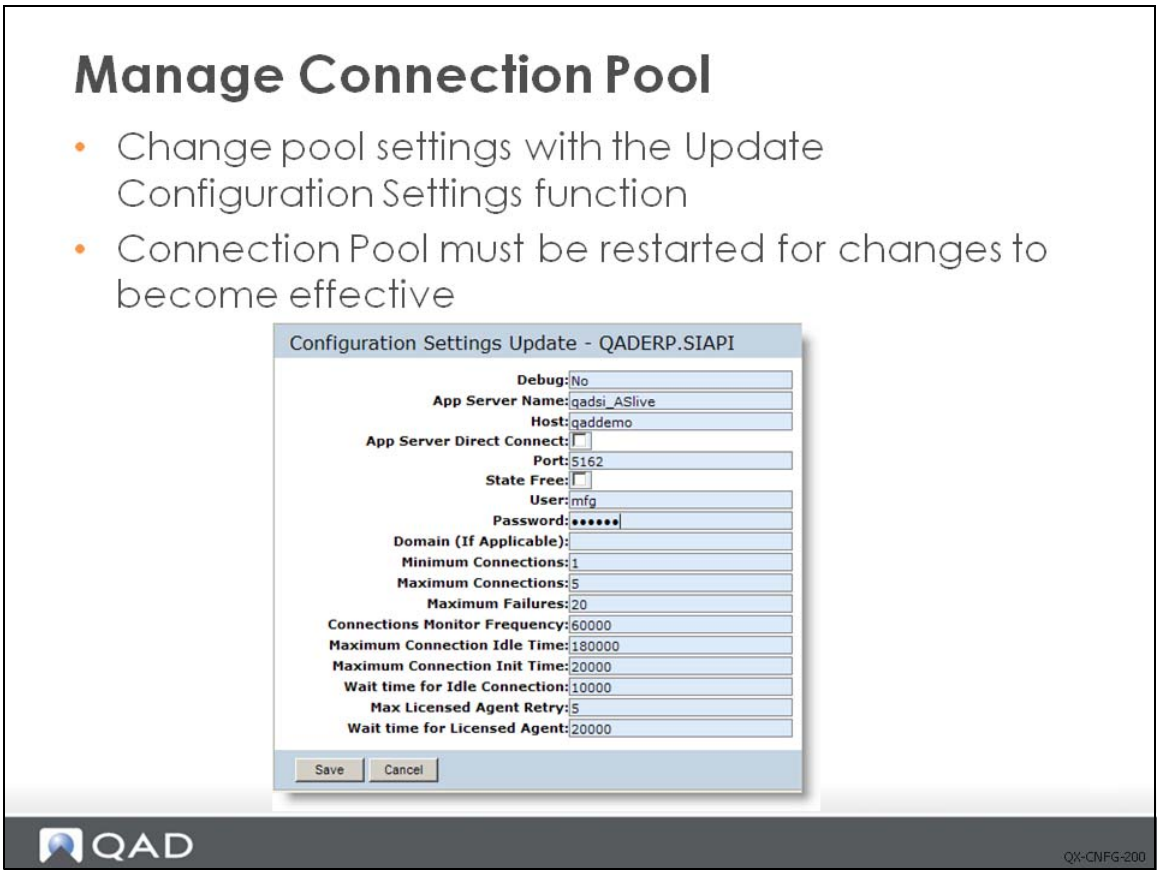


Managing Connection Pools



You can stop or start all connection pools by using the Close Connection Pool Manager, Launch Connection Pool Manager, and Restart Connection Pool Manager options from the Function node on the navigation tree menu. However, to manage an individual connection pool it is recommended that you navigate to the page for the specific connection pool and use the options that affect the current connection pool only.

To access the management page for a connection pool, navigate to the Connection Pool Manager and expand the View Connection Pool node on the tree menu. Then select the pool you want to manage from the list to display the management screen for that pool.



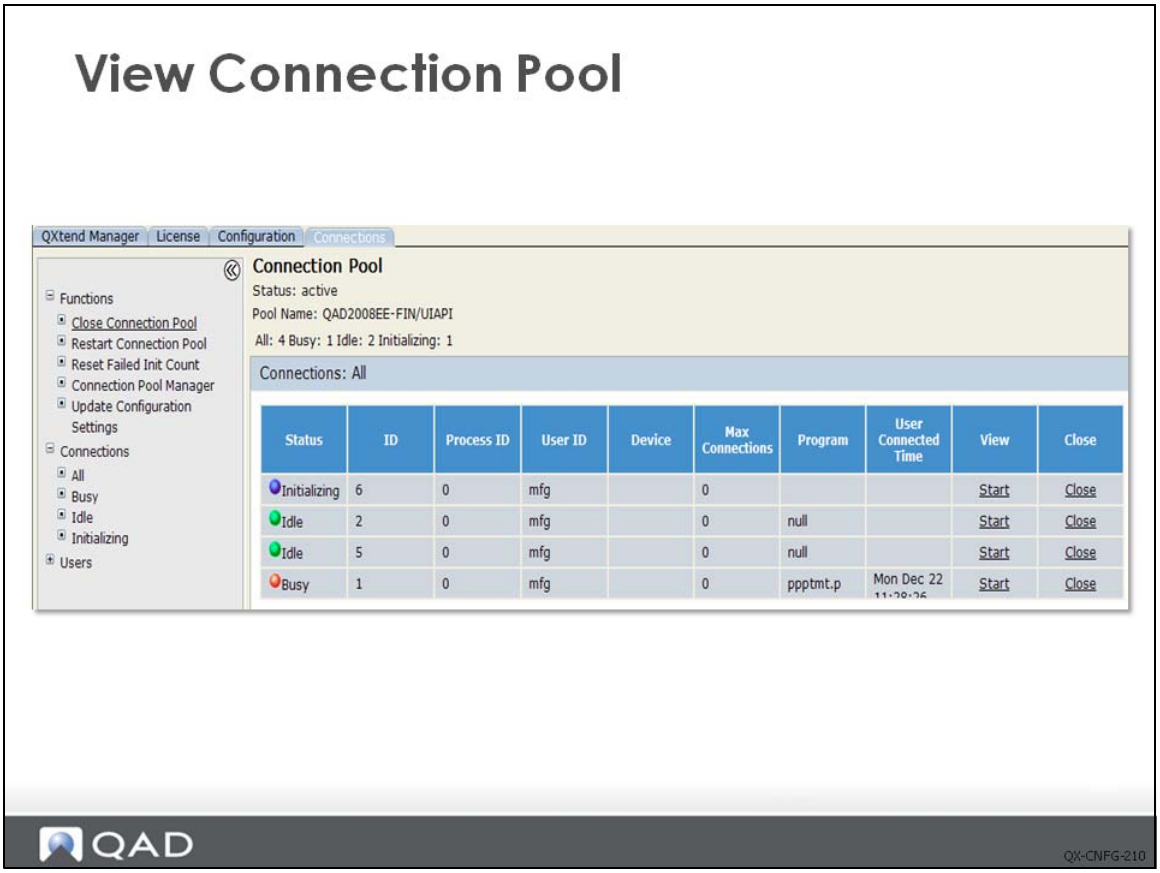
The Configuration Settings Update page on the Connections tab contains functions for managing a connection pool. The advanced user can configure the pool to ensure optimal operation, as well as adjust the size of the connection pool. To change the connection pool configuration, select Update Configuration Settings in the Functions menu. If the configuration of a connection pool is modified, the pool must be restarted for those changes to take effect since pool settings are cached when the pool is started.

The Connections node on the menu tree contains options for viewing the current status of your connection pools:

- All: Displays all connections in the pool and current status.
- Busy: Displays connections that are processing an API request.
- Idle: Displays the connections that are idle and available to process requests.
- Initializing: Displays the connections that are been started.

When setting up a new connection pool, you must ensure that idle sessions are being created before trying to process any requests to QXI.

Viewing a Connection Pool

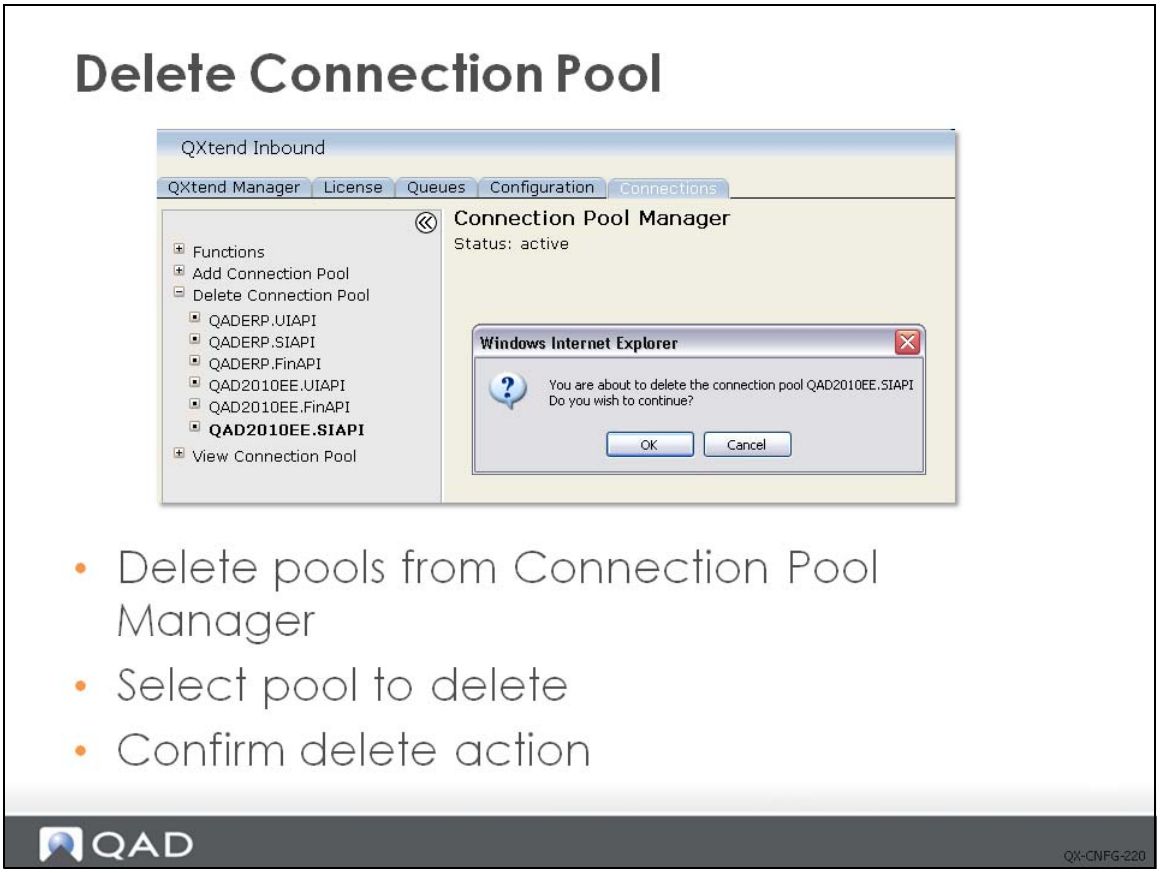


When viewing a connection pool, you can see the current status of the connection pool. Idle sessions are available to process requests; busy sessions are currently processing requests.

When a connection is busy processing a request, the program shows the API that is being executed. In the example above, the Program is set to `pptmt.p` (Item Maintenance).

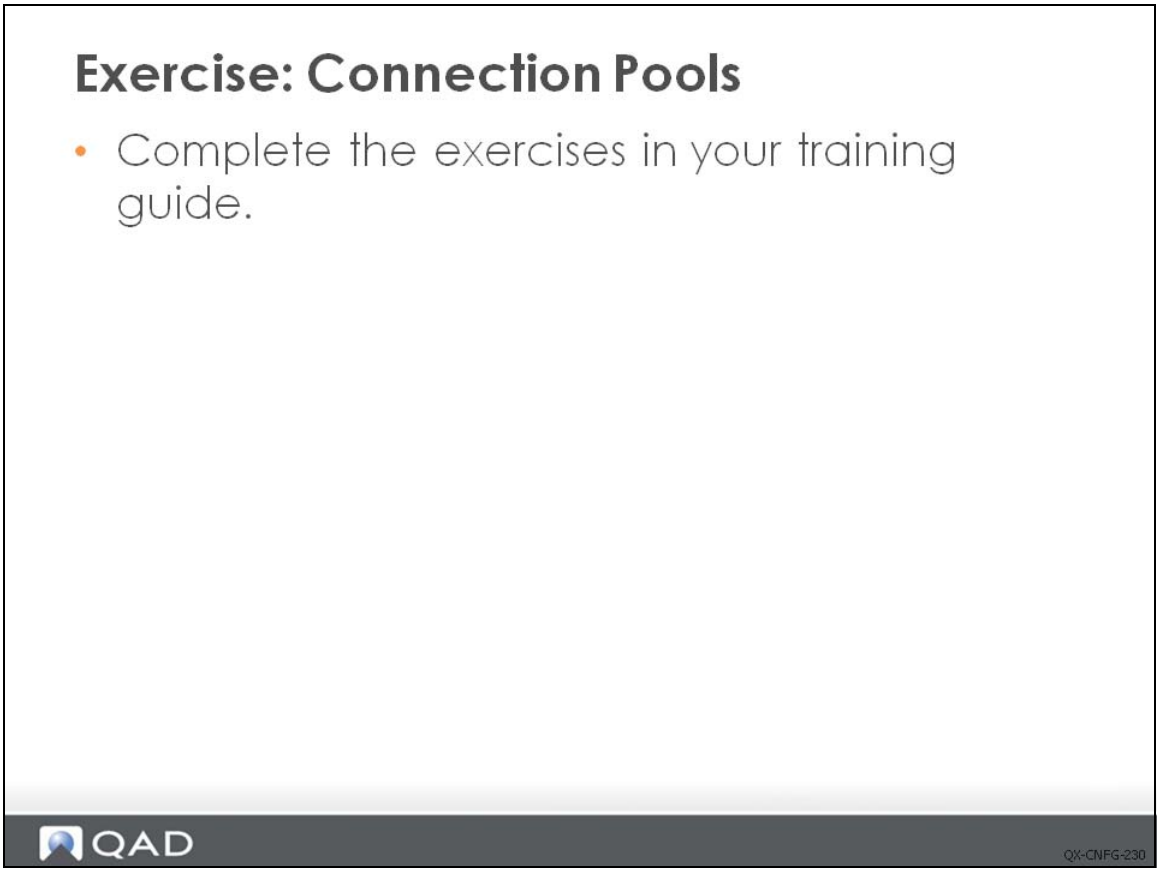
If the connection pool is a UI API pool, the View column lets you start a viewer that shows the underlying character screen used to load data into the QAD application. This is useful when debugging errors.

Deleting a Connection Pool



You can delete individual connection pools by using the Connection Pool Manager. Expand the Delete Connection Pool node to view a list of all connection pools currently configured in the QXtend instance. To delete a pool, select the pool from the list. The system prompts you to confirm the delete. When the delete is confirmed, the connection pool and all its connections are removed from the QXI configuration.

Exercise: Connection Pools



The following list shows a number of key concepts used in connection pools in QXI. In each statement below, fill in the correct term from the list.

QAD applications	pool type
pool name	UI API
connection pool name	receiver
maximum number of connections	idle sessions
request data	Connection Pool Manager
SI API	Fin API

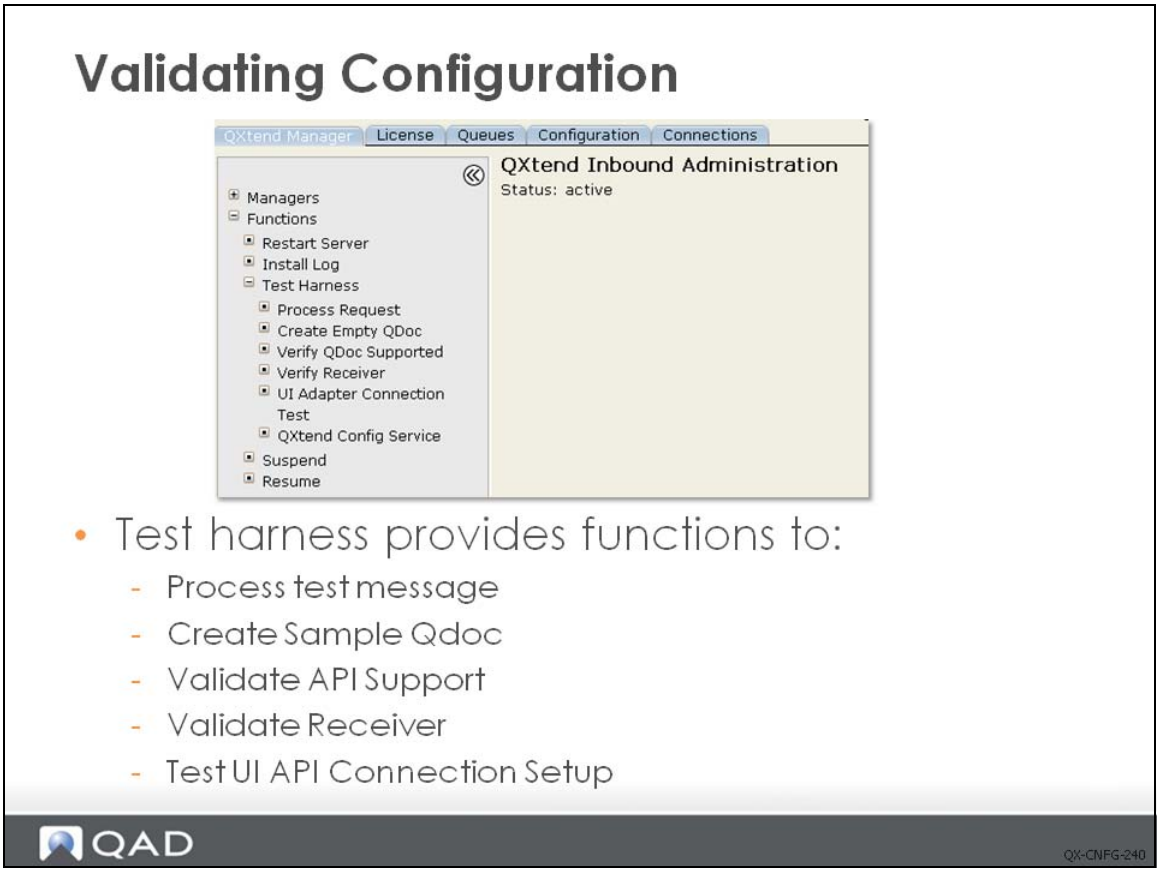
1. QXI connection pools provide connections from QXtend to specific instances of _____.
2. The connections enable _____ passed into QXtend to be loaded into the target QAD application using the APIs that are available.
3. You use the _____ to administer the connection pools available in QXI.
4. When creating a connection pool, the assigned _____ must match the name of the _____ it will process data for.
5. No new connections are created once the _____ has been reached.
6. QXI uses three types of connection pool: the _____ pool type, the _____ pool type and the _____ pool type.

7 Pool type _____ connects to the Financials AppServer.

8 A connection pool is identified by the unique combination of _____ and _____.

9 When setting up a new connection pool, you must ensure that _____ are being created before trying to process any requests to QXI.

Validating Your QXI Configuration



QXI has several functions you can use to verify your QXI configuration. Select the QXtend Manager tab, select Functions, then select Test Harness. The following options are available:

- Process Request: Lets you test the QXI web service call. To use this function you must have a complete, valid QDoc SOAP message that can be processed by QXtend.

  a Create an XML file that contains the QDoc SOAP message.

  b Copy the XML file into the directory `<tomcat>/webapps/<qxi-webapp>/WEB-INF/receivers/<receiver-name>/requests` on the Tomcat server that is running the QXI instance. The request is placed in a directory under a specific receiver.

  c Select the “Process Request” menu option and provide the name of the xml file and the name of the receiver, click “Submit”

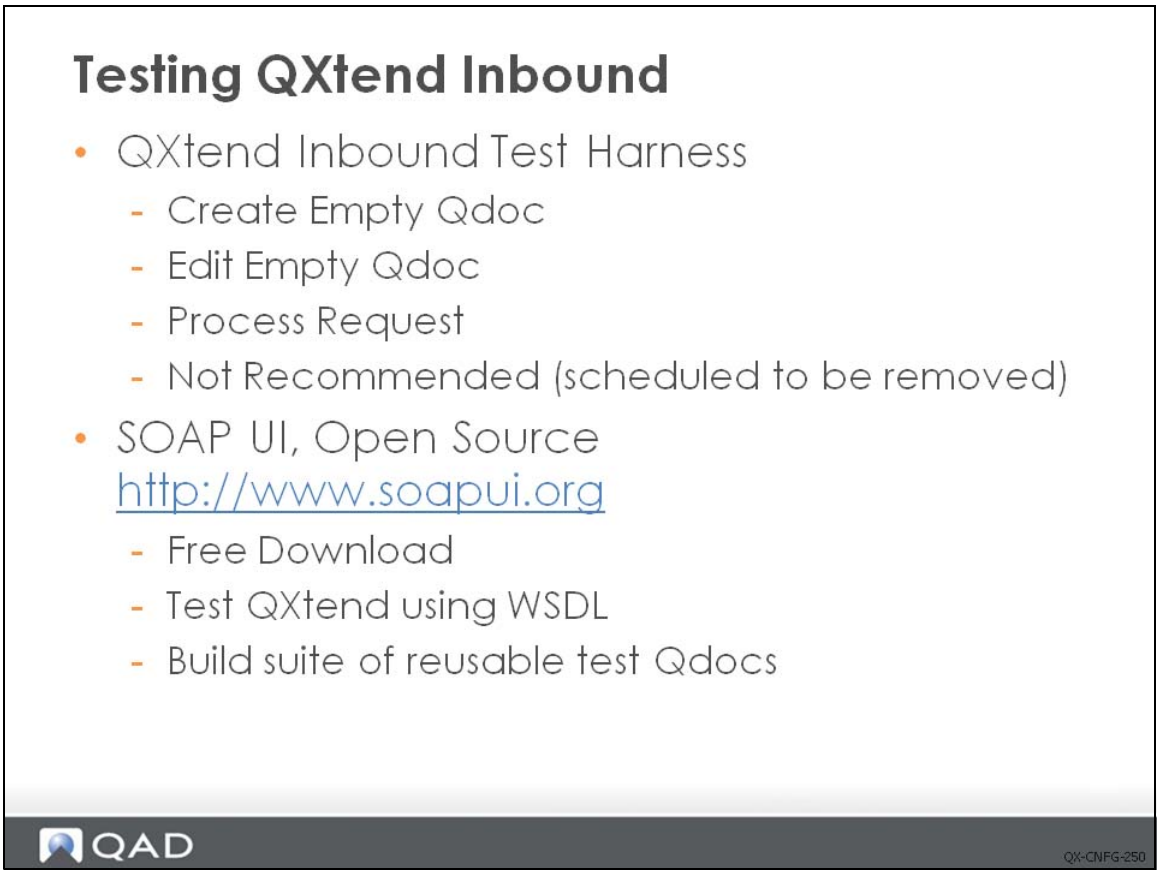
  d The response message displays and be analyzed for any errors.

  This function has been in QXI since its inception but will be removed in future releases. Now there are user-friendly open source tools that can be used to test the product. It is recommended that you use the free product SOAP UI that can be downloaded at `www.soapui.org`.

  The labs in this class will use SOAP UI, not the Test Harness, to process requests.

- Create Empty QDoc: Creates a sample QDoc request for the selected QXtend API. It uses the API schema and builds a sample request. The generated request will not process until the data in the QDoc is set up to meet the requirements of the target QAD application. Sample requests are generated in `<tomcat>/webapps/<qxi-webapp>/test`. This function also can be replaced with SOAP UI functionality.
- Verify QDoc Supported: Used to test whether or not the specified API and API version are supported against the specified receiver. This function can be used to validate the receiver configuration to determine if an API is available to be used against a receiver. This check is performed against every request that is processed by the QXtend Web service.
- Verify Receiver: Used to validate that the entered receiver is set up in the QXtend instance.
- UI Adapter Connection Test: Used to process a dummy request to ensure that the UI adapter connection is configured correctly. The UI API adapter leverages a telnet connection to load data into QAD applications. The UI Adapter Connection Test function uses the UI API connection pool set up against a receiver. When the connection pool is created, success is indicated simply by having a connection pool that has idle connections. However, this does not necessarily mean that the setup is correct. To verify the setup, run the connection test. This test should be performed every time a new UI API connection pool is created.

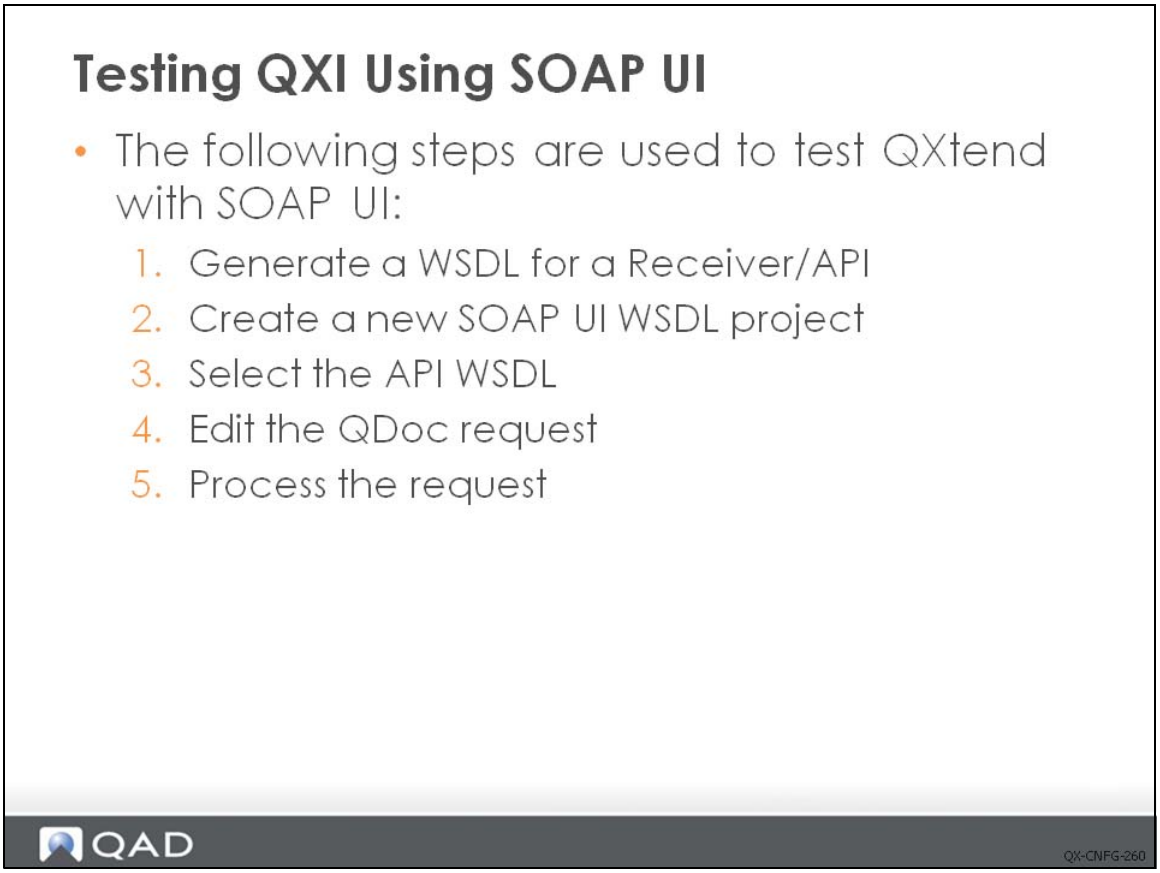
  Enter the receiver and (if applicable) domain to test the connection, then click Submit. The response will be Success if the setup is correct.



You can process test requests using QXI in several ways; the test harness is functionality that is currently built into QXtend. As the test harness does not provide a consolidated environment for testing QXtend—and because there are open source tools available that provide a better environment in which to build ad test QDocs—this functionality will be removed in future releases of QXtend.

The recommended testing approach for QXtend is to use SOAP UI. SOAP UI provides a Web Service testing environment that lets you build QDocs and test the QXtend Web Service easily. You can download a free version from `http://www.soapui.org`. All manual testing in this class will use SOAP UI.

Testing QXI Using SOAP UI



Testing the QXI Web Service using SOAP UI is not a complex task. SOAP UI allows you to save your test cases into a project and re-use them to test QXtend. Some features allow you to run test cases as a test suite.

To test the QXI Web Service, do the following:

1 Generate WSDLs for a Receiver

SOAP UI is a web service testing tool and requires a WSDL document that describes the API to be available. WSDL files describe the API Interface, they also describe where the service is and how to access the service. The API definition is split into 3 components:

- WSDL: describes the entire API, the request and the response.
- Request Schema: describes the request data that can be sent to the API you can think of this as the data that gets entered.
- Response Schema: describes the data that is returned after a request has been processed this can include Invoice and Sales Order numbers that were created during the processing of the request.

Generate WSDL button is available in Modify and View Receiver screen. Select the APIs which you want to generate a WSDL, click Generate WSDL, save and unzip the file.

2 Create a New SOAP UI WSDL Project

SOAP UI is a project based tool and provides an environment to build Web Service testing projects. Ensure that the “create sample requests for all operations” option is selected and this way when the project is created a sample request is automatically created.

3 Select the API WSDL

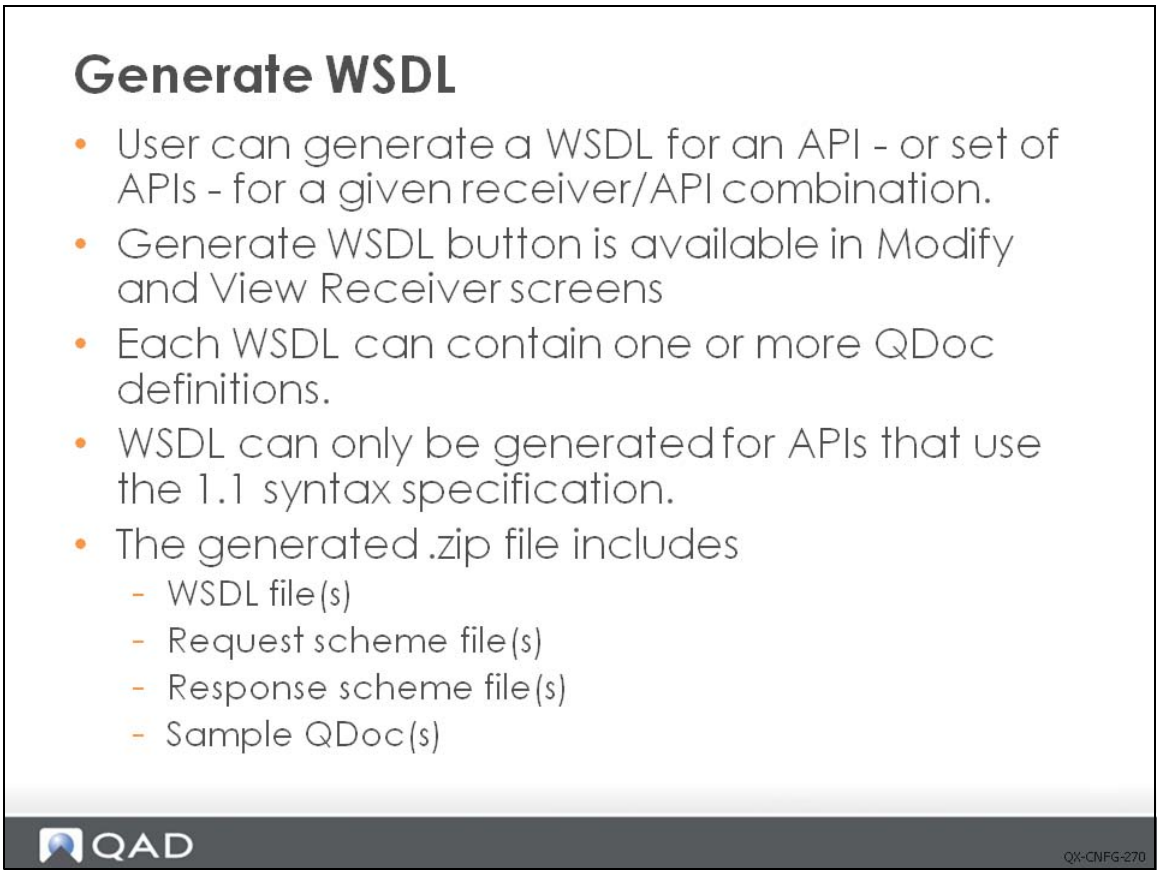
When you create a new project you are prompted for the WSDL file to create the project from, this will import the operations defined in the WSDL and create sample requests. Browse to the location where you unzip the zip file and select the WSDL for the API file you want to test.

4 Edit Qdoc Request

Once the project has been created, a sample Qdoc is generated. Because the QDoc is generated without any knowledge of QAD QXtend or the target QAD application, the QDoc does not contain any application data and needs to be edited before calling QAD QXtend.

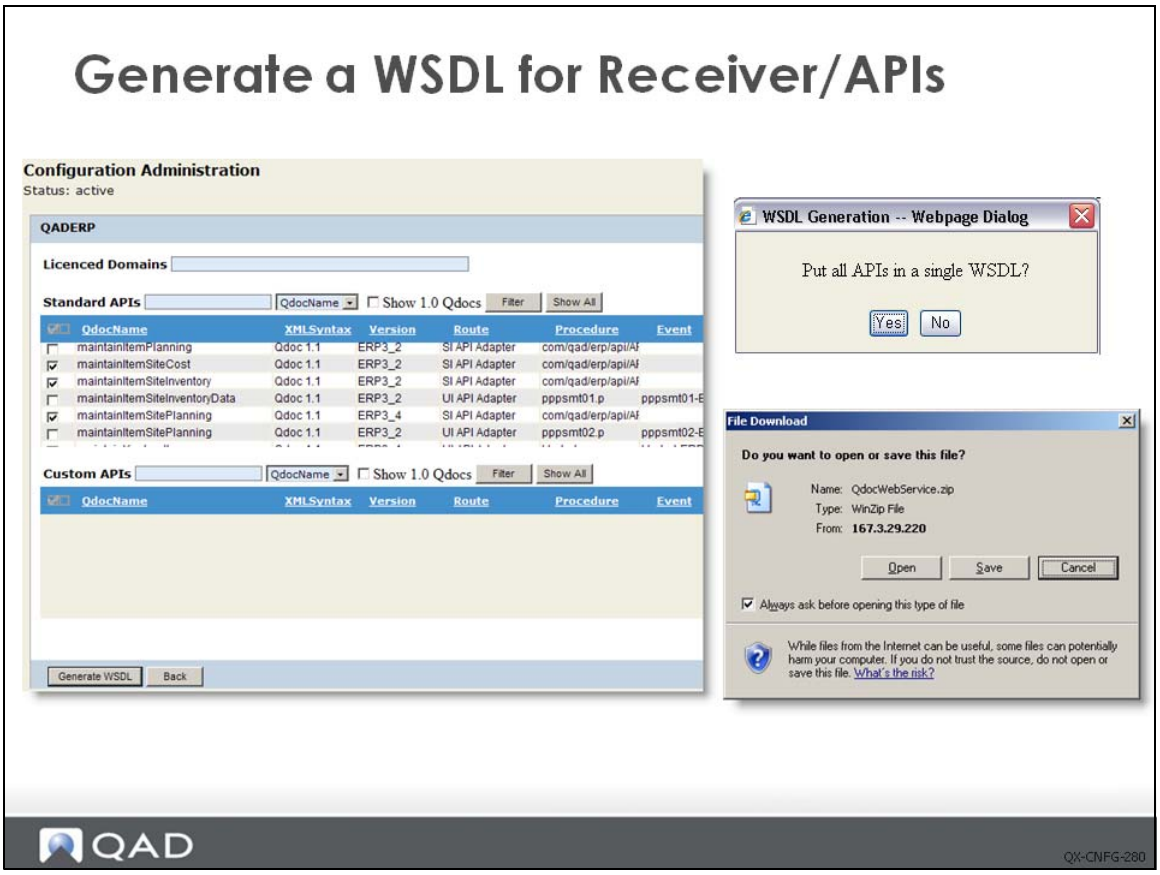
5 Process Request

Once the project has been created and the request is updated, the final step is to process the request. The green arrow in SOAP UI submits the request to the web service and displays the response.



You can create a WSDL on demand for APIs. Each WSDL can contain more than one QDoc definition, allowing multiple APIs to be combined into one WSDL file for a selected receiver. All the request and response schemas are included and can be saved to a client. To generate a WSDL, select a receiver and then click the Generate WSDL button.

If a WSDL is generated from a single API, both the compressed .zip file and its contained WSDL .xsd file use the same name as the API name. If a WSDL is generated from multiple APIs, the zip file is called `QdocWebService.zip` and WSDL file is called `QdocWebService.wsdl`.

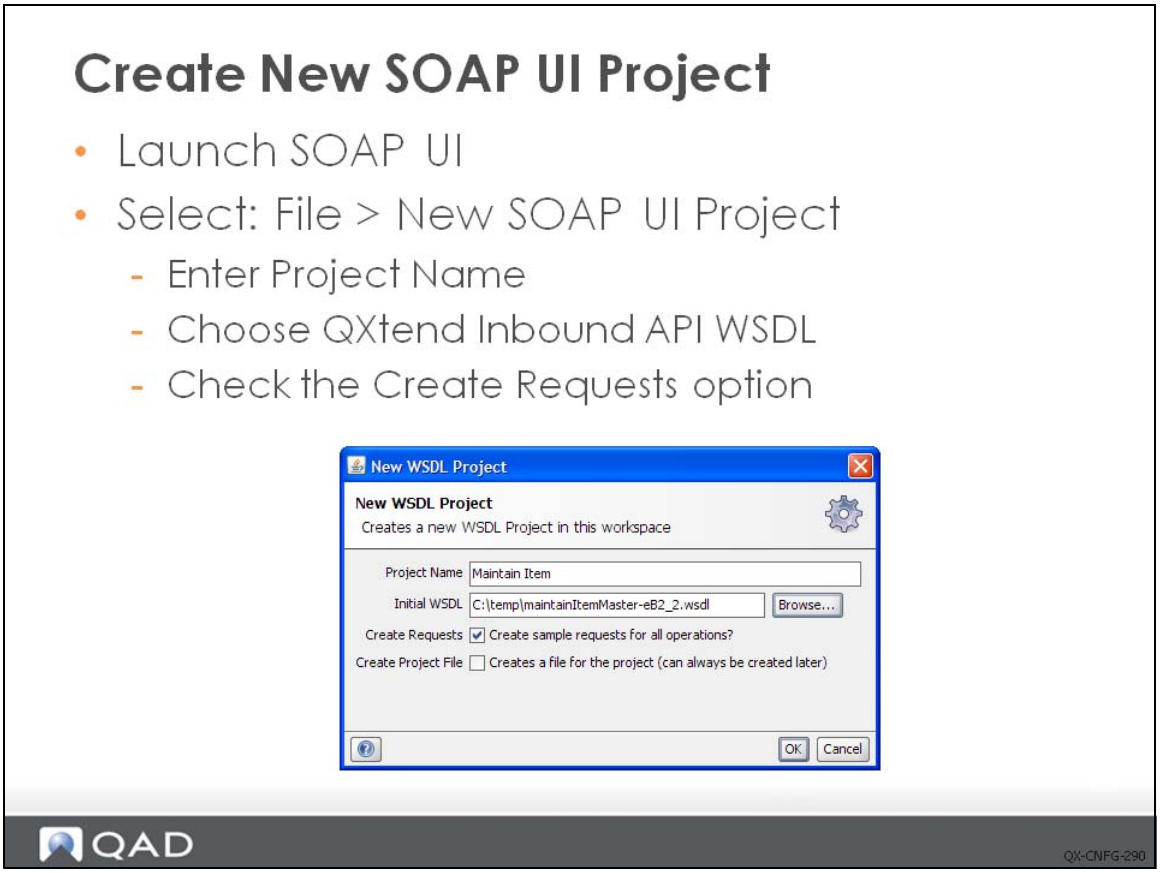


To generate WSDL:

1. In the Configuration Manager, choose Receivers|<application>, where <application> is eB, eB2, eB2.1, QADSE, or QADEE. Other products also might display on this menu. The currently available receivers for that version display.
2. Select the receiver to generate a WSDL for by selecting its check box and then click View.
3. A list displays of the standard and custom APIs that are currently supported by the receiver.
4. Select the APIs for which you want to generate a WSDL. You can select any combination of standard APIs, custom APIs, or both.
5. Click Generate WSDL. Note the following:
   - If no API is selected, a warning message appears when you click Generate WSDL. Exit the message box and select an API.
   - If you select only one API, you have the option to generate a backward-compatible WSDL that can be used with an older version of an API. You also can generate a backward-compatible WSDL if you select multiple APIs but choose not to combine them in a single WSDL.
   - If you select multiple APIs, you have the option of combining all the APIs in a single WSDL. If you select No, you have the option to create a backward-compatible WSDL.
6. The WSDL is generated and a File Download box displays.

7 Navigate to where you want to download the file and click Save. The .zip file is saved to the selected location.

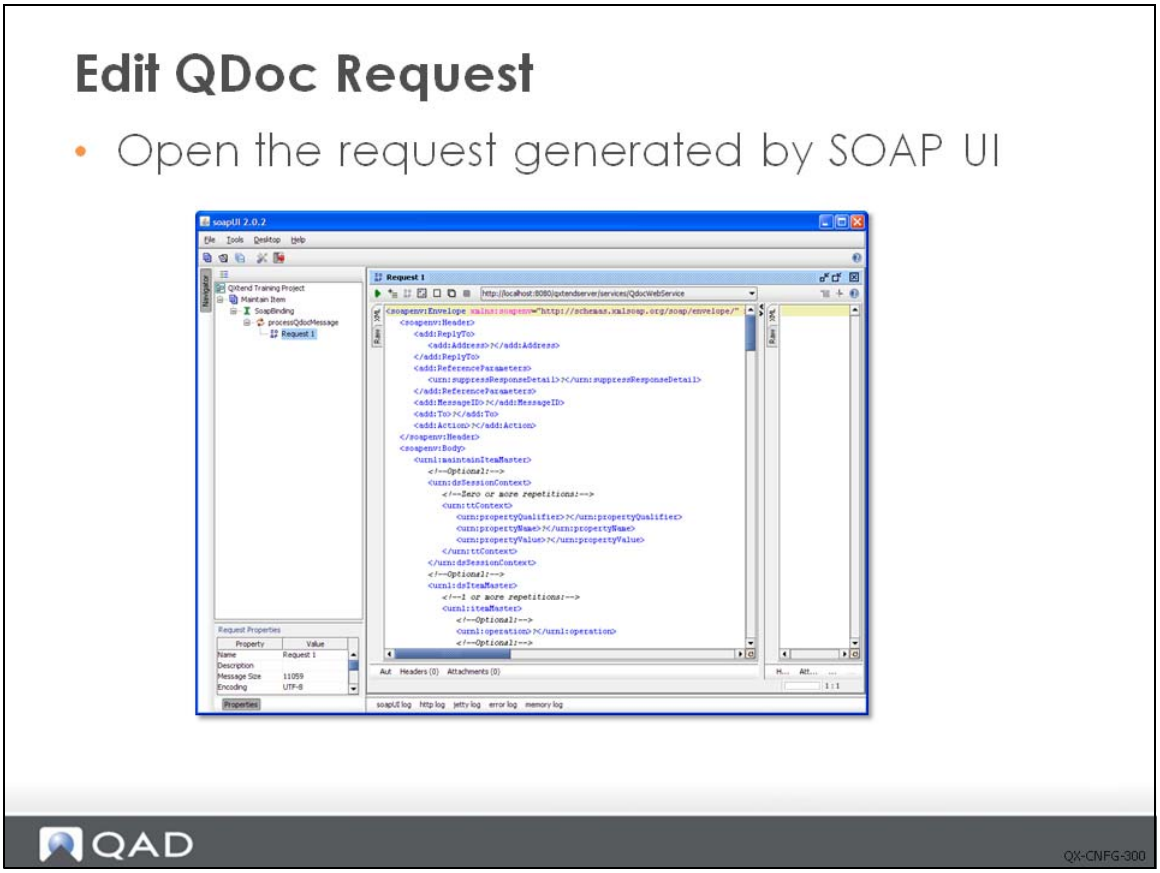
Create New SOAP UI Project



Select File|New WSDL Project to create a new project from a QXI API WSDL document. Enter the project name and select the WSDL document to use. Ensure that the Create Requests option is selected so that when the project is created, SOAP UI will create a sample request for the `processQdocMessage` operation.

Each API that needs to be tested within SOAP UI requires a separate project because QXI supports a generic message-based Web Service rather than the RPC-style Web Service.

Edit QDoc Request



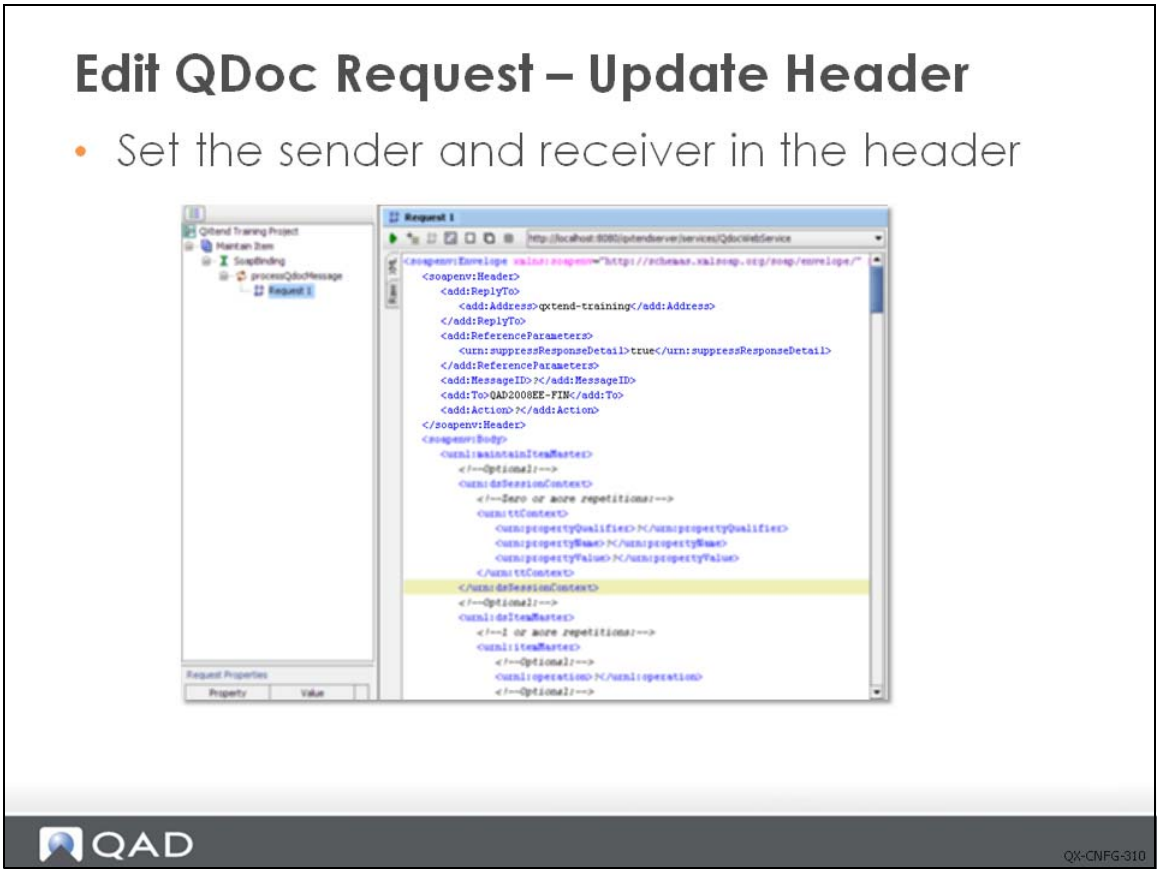
The sample request that is generated by SOAP UI interprets the definition contained in the QXtend API WSDL and schema, and generates a SOAP message that includes the elements that can be included in a request to the service. The request does not contain data that is required by QXtend or the target QAD application, and the request must be edited before it can be successfully processed.

To access the sample request, expand the project in the Navigation pane, expand the SoapBinding option, expand the processQdocMessage option, then double-click the Request 1 entry. The request opens in Edit mode.

SOAP UI provides has some useful features for editing requests:

- Format XML: Right-click the request during data input and select Format XML. SOAP UI automatically formats the request.
- Validate: After entering the request data, right-click and select Validate. SOAP UI validates the data entered into the request against the QXtend API WSDL and schema, and reports any exceptions such as invalid date, number, and logical or missing mandatory data.

Edit QDoc Request (Update Header)



A QDoc SOAP message has three sections:

- SOAP header
- SOAP body – session context
- SOAP body – request data

Updates are required to each of these sections to ensure that QXI can successfully process the request.

The SOAP header contains information that QXI requires, such as addressing information and processing directives that control how QXtend processes the request. The 3 most important elements are:

- Sender: Used to identify the source of the request. Typically the sender is the application instance that sent the data to QXtend. QXO is a good example of a sending application. The sender is defined in the `ReplyTo` node of the SOAP header.

  ```
  <add:ReplyTo>
  <add:Address>qxtend-training</add:Address>
  </add:ReplyTo>
  ```

- Receiver: Used to identify the receiver in QXI that should be used to deliver the request to the target QAD application; this element is the most important element in the SOAP header. The receiver name is the user-defined name assigned to identify an instance of a target QAD application as discussed earlier. The value defined here controls where QXI loads the request data to. The receiver is specified in the `To` element in the header.

  ```
  <add:To>QADERP-FIN</add:To>
  ```

- Suppress Response Details: This element is optional. If it is not supplied in the request, QXtend defaults it to `true`. This element controls the level of detail returned to the calling application in the response message.

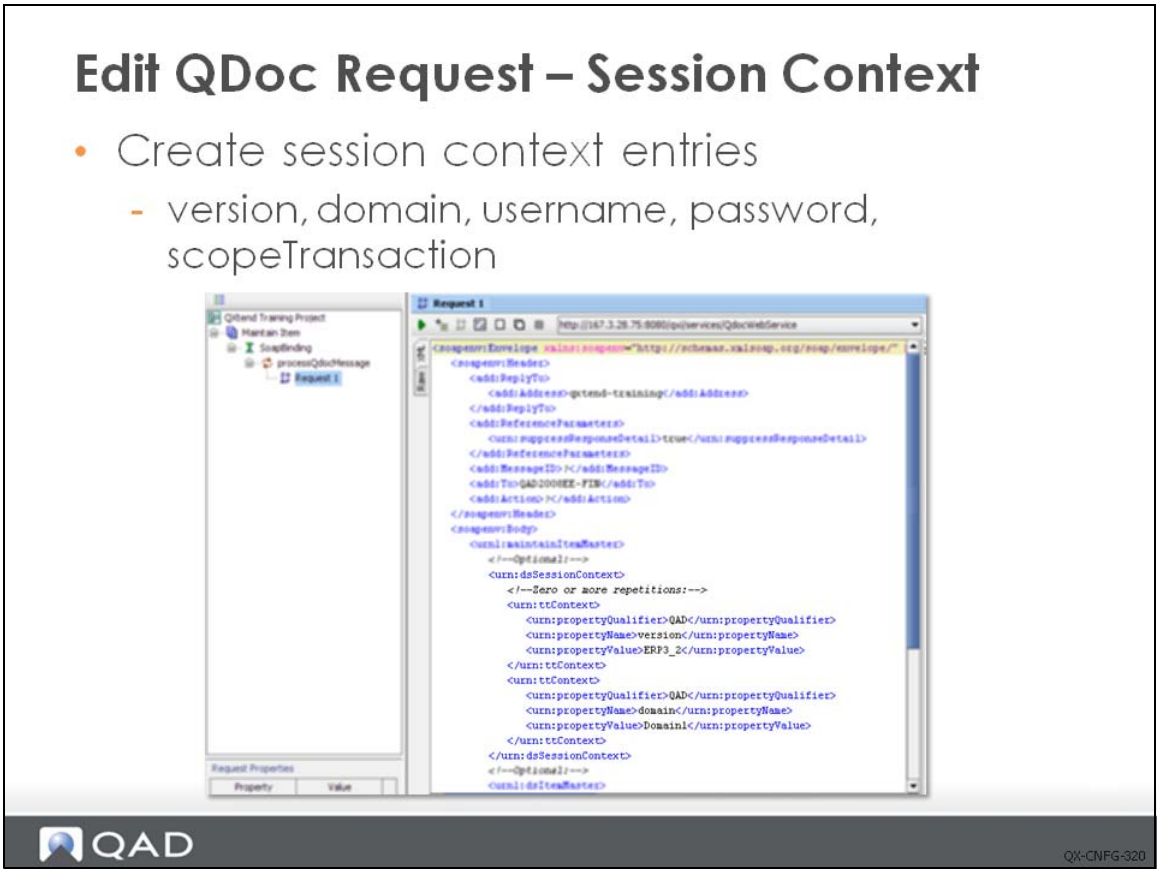
  When `true`, the response message from QXI will only contain an indication of the processing result (success, error, warning). If there are any error/warning messages raised during the processing of the request, these are also returned.

  When `false`, the response message from QXI contains more details; the processing result and any errors are returned along with the data defined in the response schema. The response schema for an API typically returns the values of the keys of the updated records, but the schema can be configured to return whatever data is required. The response schema data is returned only when `suppressResponseDetails` is set to false.

  The setting for this is part of the `ReferenceParameters` node:

  ```
  <add:ReferenceParameters>
  <urn:suppressResponseDetail>true</urn:suppressResponseDetail>
  </add:ReferenceParameters>
  ```

Edit QDoc Request (Session Context)



The body of the QDoc SOAP message has two sections:

- Session context
- Application data

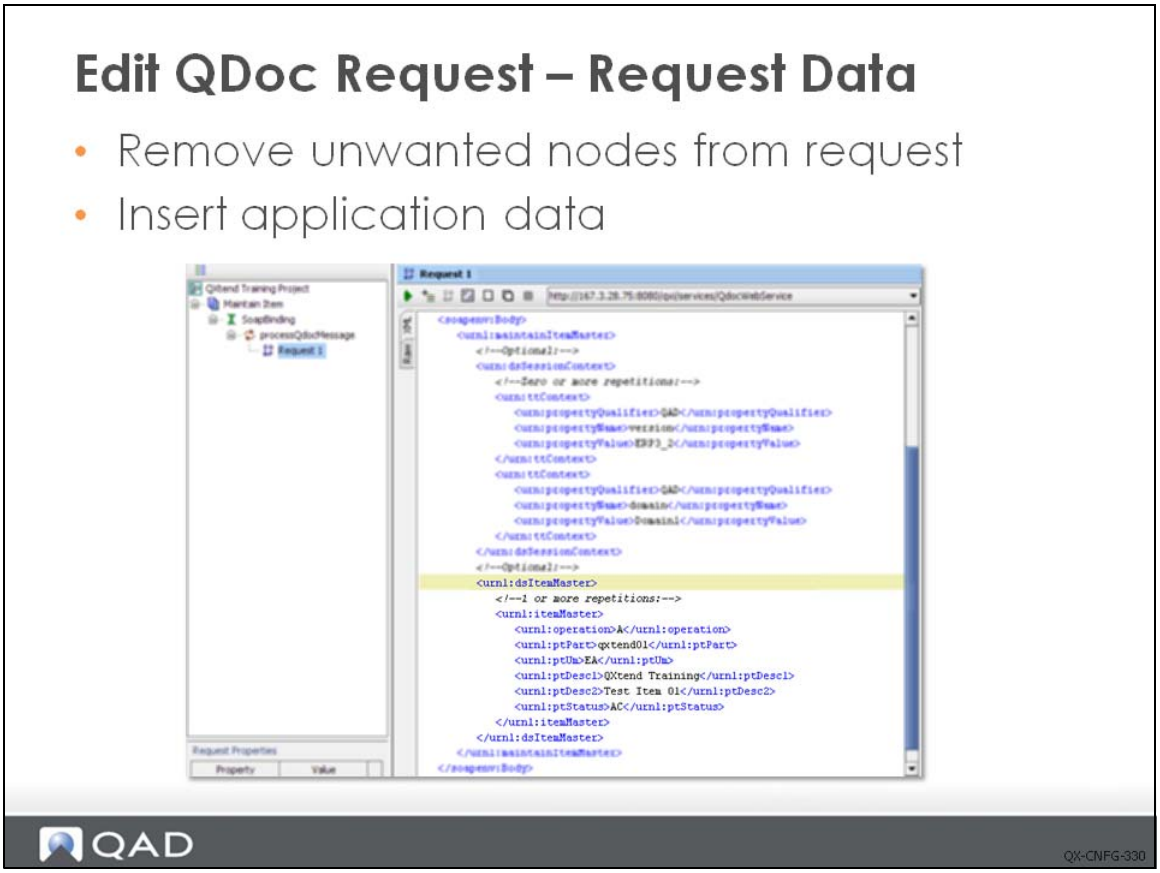
The data in the `session context` section controls how the request is processed both within QXtend and in the target QAD application. Several valid parameters can be sent, some mandatory and some optional. The following parameters are supported and represented using the following format:

<propertyQualifier>/<propertyName> – <parameter description>

- `QAD/domain`: An optional parameter that identifies the domain the data should be loaded into in the target QAD application. This parameter is required only when the target application supports the QAD domain architecture. If this value is not supplied, the domain defined on the connection pool is used.
- `QAD/version`: A mandatory parameter used to determine the version of the API to use to process the request; for example, `eB2_2`. The combination of the QDoc name and version determine the API adapter/connection pool used to process the request.
- `QAD/scopeTransaction`: An optional parameter that defaults to `false` if not included in the request. This parameter controls the transaction scope during the processing of a request with QXI. If set to `true`, the entire QDoc is treated as a single transaction; if any error messages are received the entire transaction is backed out. If set to `false`, the traditional transaction scoping of the underlying QAD application is used.

- `QAD/mnemonicsRaw`: An optional parameter that allows the user to define whether or not fields in the user interface that use language mnemonics should be entered using the universal code value or the translated string that is dependent on the setup of the user processing the request. If set to `true`, fields that use mnemonic values will only accept the mnemonic code, and not the value that is normally displayed on screen.
- `QAD/username`: An optional parameter that allows the user name to be used to process the request when the Use Requestor feature is enabled with QXtend; otherwise the user name defined on the connection pool is used. For details on Use Requestor, see *Technical Reference: QAD QXtend*.
- `QAD/password`: An optional parameter required when a `username` has been defined in the request. The password is required for the `username` parameter to ensure that the session can log in. The username and password are required if the Require Authentication attribute of the receiver is set to true.
- `QAD/sessionID:` If a valid QAD Enterprise Application session ID is provided in the QDoc, the QDoc can pass authentication without having the username and password checked. The system also caches and reuses the session ID provided in the response QDoc to further speed up authentication and improve performance. If the session ID is no longer valid, the system checks the username and password.
- `QAD/action`: An optional parameter only used with QAD EE, this parameter defines the action being performed by the request. This parameter must be used with functions that leverage the new infrastructure that drives the Enterprise Financials in the Enterprise Edition. The typical value when synchronizing data between instances of EE is `Save`.
- `QAD/entity`: An optional parameter only used with QAD EE, this parameter defines the entity in QAD EE that the data is to be loaded into. If a value is not supplied, the default entity for the user set up against the connection pool is used.
- `QAD/email`: Email address(es) of person requiring notification of the message after receiver has processed it.
- `QAD/emailLevel`: Email delivery level (1 or ERR - send on error, 2 or WRN - send on error or warning, 3 or MES - send all).

Edit QDoc Request (Request Data)



The `request data` section of the SOAP–BODY contains the actual application data that needs to be loaded into the target QAD application.

When a new WSDL project is created in SOAP UI, by default it includes every XML node defined in the API interface definition: in the case of QXtend, in the QDoc XML schema. This means that if you create a project for a Sales Order, the sample message will contain hundreds of XML nodes/fields—and iterations such as Comments, Tax, and so on—that may not be required.

The first task when editing the request data is to identify the nodes/fields required to successfully process the transaction you are trying to load into the QAD application. The best approach is to only supply the values in the request that are definitely required, as this will help to minimize the amount of data being passed across the network.

Once the sample request has been trimmed down to the required elements, the next step is to update the request with the application data that you want to load. The nodes in the XML are named to represent the data elements that will be loaded: for example `<soNbr></soNbr>` represents the location in the message for the sales order number. Update the nodes with the application data; for example, `<soNbr>SO12345</soNbr>` defines a sales order number of SO12345. When processed this value will be used to update the existing order, or to create a new one with the specified number. All nodes in the request should be updated to contain data that needs to be loaded.

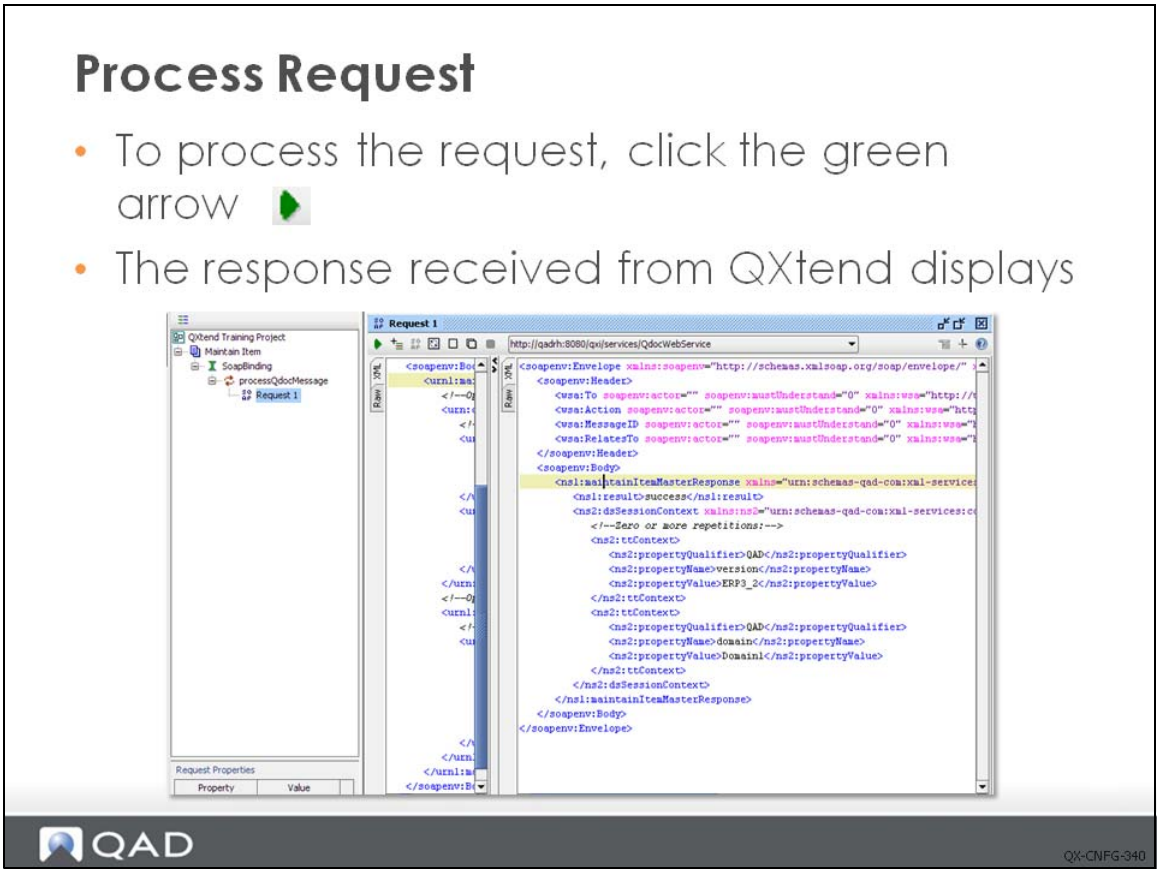
In the example, a new item is being created with the item number = `qxtend001`, unit of measure = `EA`, Description 1 = `QXtend Training`, Description 2 = `Test Item 01`, and Status = `AC`. Other values not defined in the request are assigned default values.

Additional Notes

The different adapters supported by QXtend behave differently when values are removed/omitted from the XML. The following list describes how each adapter handles removed nodes.

- UI API adapter: When adding, fields are assigned the default value configured within the application. When modifying existing data, the value is left unchanged.
- Fin API adapter: When adding, fields are assigned the default value configured within the application. When modifying existing data, the value is cleared and any data previously stored in the field is lost. When modifying a record, ensure that all data for the financial component is included in the request, otherwise unexpected errors may occur when processing the request resulting in lost data.
- SI API adapter: When adding, fields are assigned the default value configured within the application. When modifying existing data, the value is left unchanged.

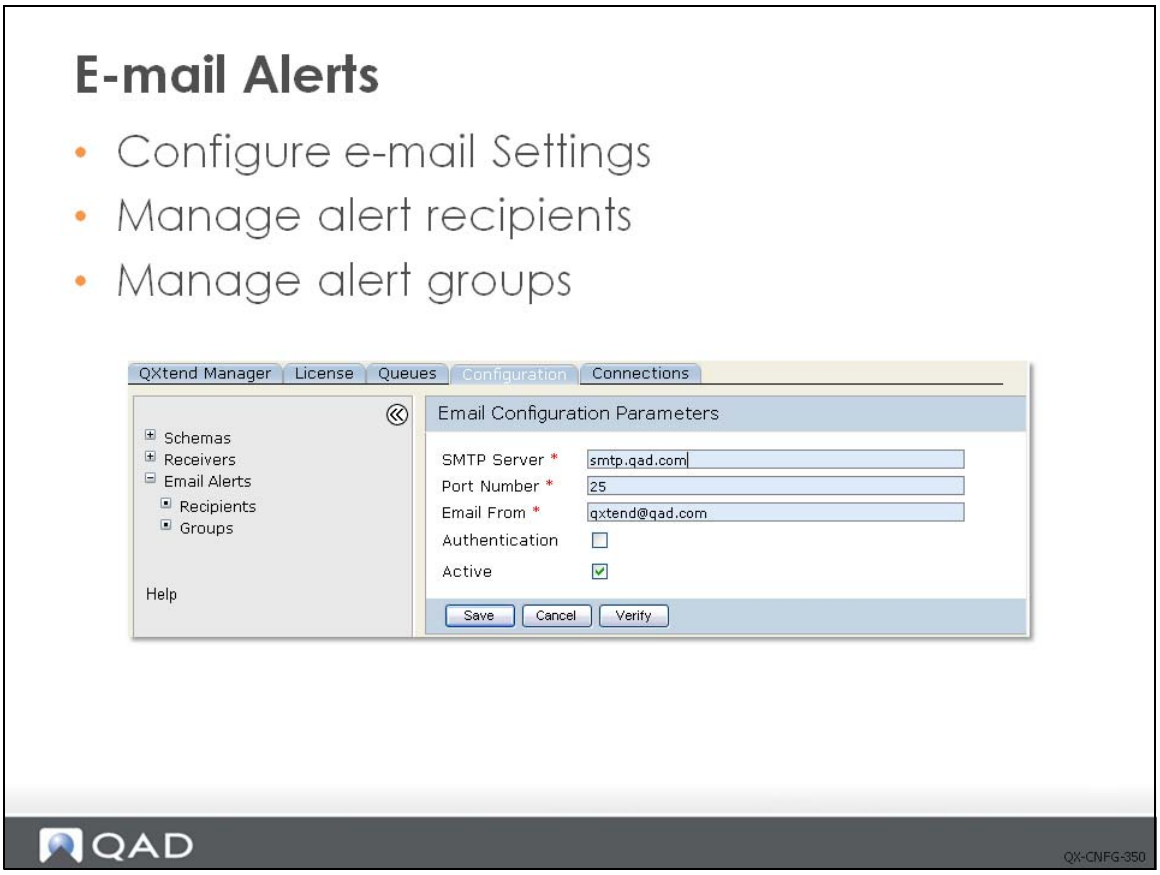
Process Request



The final step is to process the request you have built by invoking the QXtend WebService, and passing the QDoc XML message from SOAP UI to QXI. The green arrow icon in the Request window executes the Web Service call and passes the XML to QXtend.

The request is processed and SOAP UI waits for the response from QXI. When the response is received it display in the response section of the Request Message window. The response details indicate the status of the response; the `<ns1:result></ns1:result>` node contains the status, which should be success, error, or warning.

Errors that occur during message processing display in the `dsExceptions` area of the response message.



The e-mail alerts feature allows specified recipients to receive e-mail alerts that are raised by QXI for events relating to specific receivers, domains, and APIs.

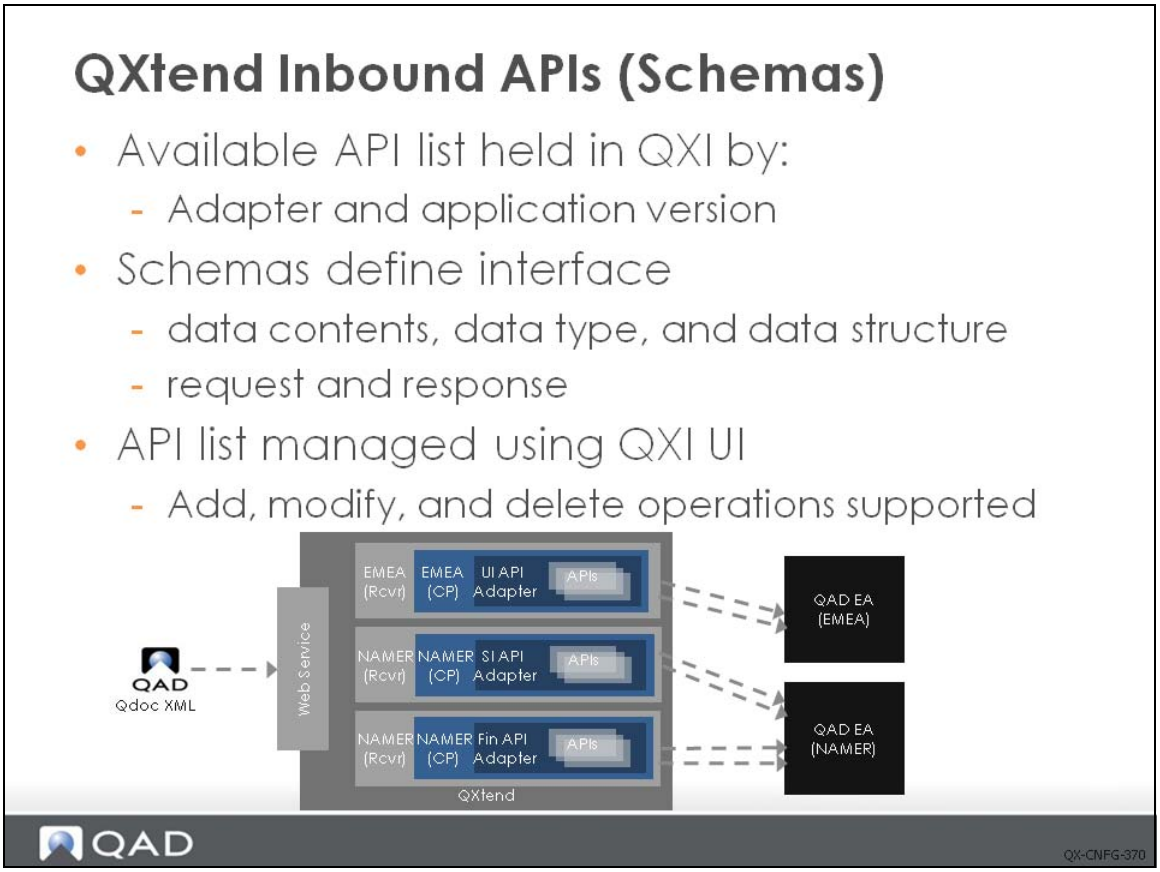
Alerts in QXI are generated by two types of events: Default (system) alerts and Processing alerts.

In the QXI Configuration Manager you can:

- Configure e-mail settings.
- Manage alert recipients.
- Manage alert groups.

For details on E-mail Alerts, see *Technical Reference: QAD QXtend*.

QXI APIs (Schemas)



QXI provides access to many APIs; the list of available APIs is maintained in QXtend under the Schemas node on the menu tree on the Configuration tab. QXI allows APIs to be added, modified (only certain details), or deleted.

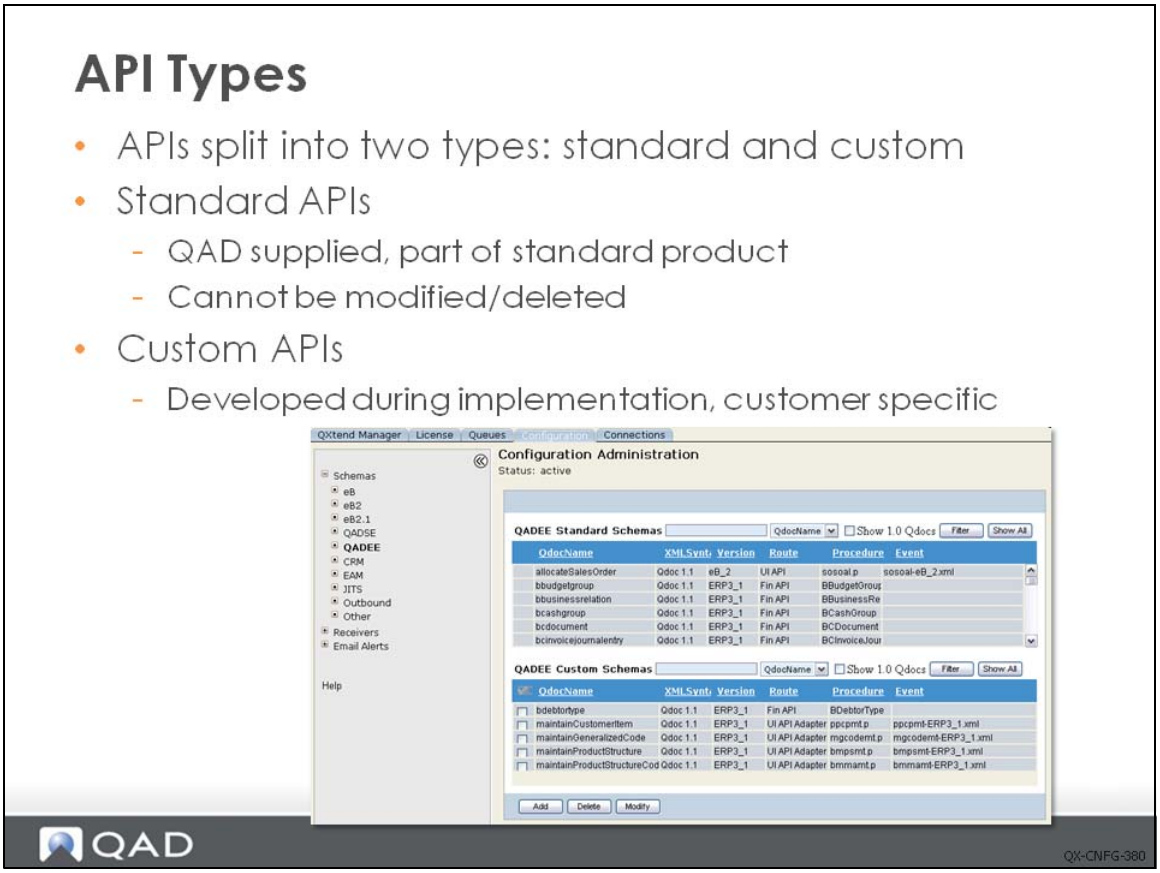
The schemas define the API. An API has two schemas; one schema defines the content of the request message, the other defines the content of the response message.

API names have two components `<apiName>-<apiVersion>`; the combination of `apiName` and `apiVersion` combine to create a unique identifier for the API. A specific version of an API is linked directly to an adapter; QXtend maintains the list of APIs that are valid for an adapter.

Example If maintainItem-ERP3_2 is linked to a UI API Adapter, when a maintainItem SI API is created, the new API would be called maintainItem-ERP3_3.

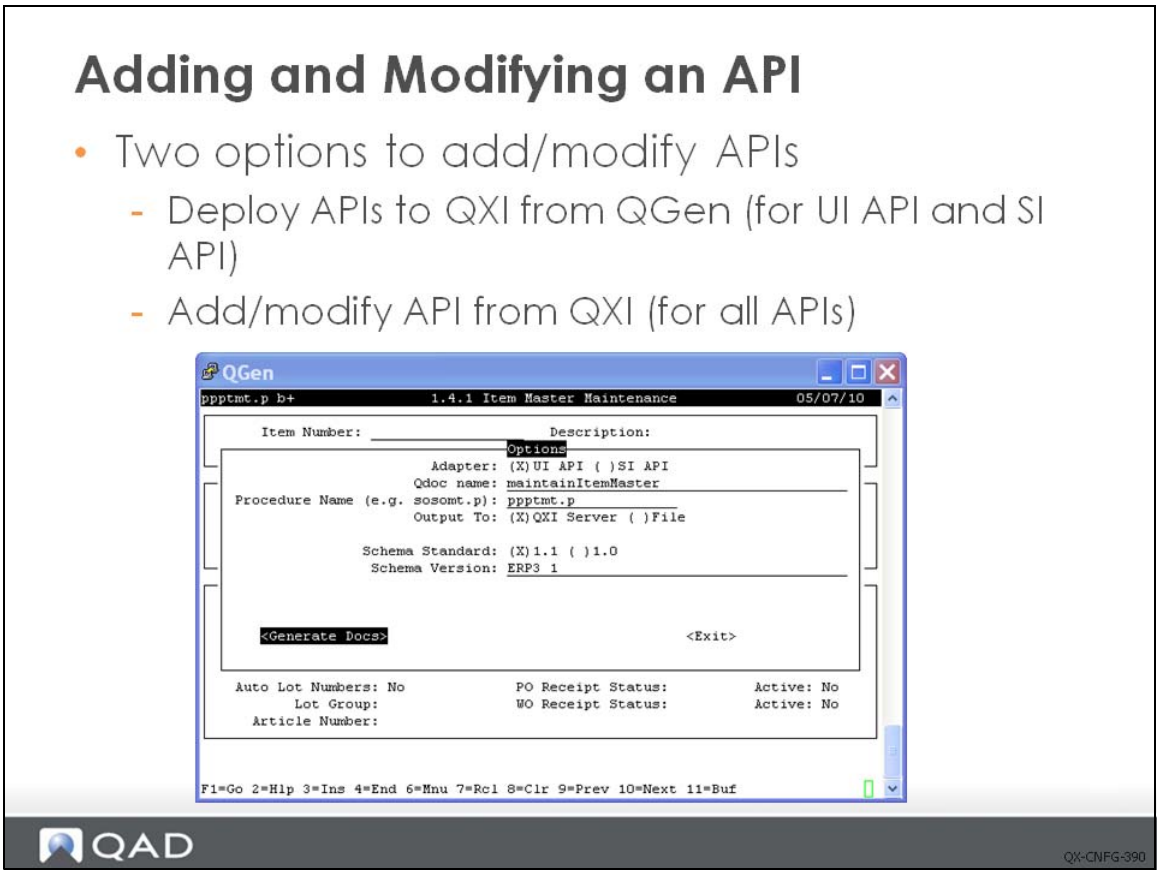
APIs are only valid for the QAD application they were developed for—for example, an API used in QAD EAM is valid only when the receiver is a QAD EAM receiver. When APIs are added to QXtend, they are added for a specific application type and can only be used on receivers that are linked to the same application type.

API Types



APIs in QXI can be grouped into two types: standard and custom.

- Standard APIs are either shipped with QXtend, or are supplied later by QAD as part of future QXtend releases. These APIs support the standard functions and are not designed to work with customizations made to the QAD application. Standard APIs cannot be modified or deleted as they are controlled by QAD. When new APIs are added to QXI, you have the option as adding them as either standard or custom.
- Custom APIs allow for APIs that are developed to support the QXtend implementation. These APIs are required to support customizations that have been made to screens during the application implementation, or to provide APIs that are not part of the standard QXtend release. If a standard API is modified and loaded as a custom API, the version number used for the API must be different to the version number of the corresponding standard API. Custom APIs can be modified and deleted once they have been added to QXI.



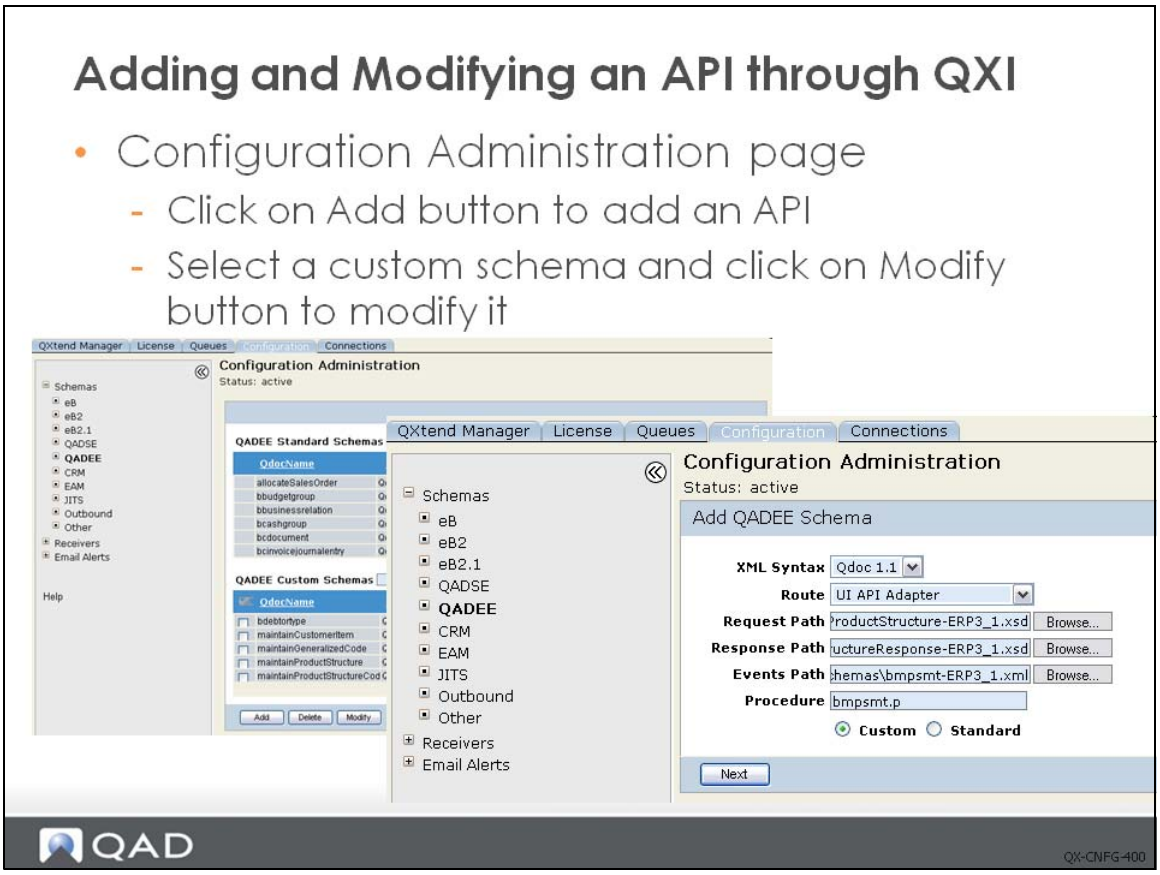
An API can be added/modified in two ways: using QGen to deploy the API, or using QXI to add or modify an API. Using QGen is the recommended way as API can be generated from QGen and deployed to QXI seamlessly.

QGen is a development tool that facilitates the creation and maintenance of QXtend APIs. The UI API adapter uses the character screen within the application to load data. QGen records the structure and field information from QAD EA character UI screens and saves the information to data files. QGen can also generate API required files (schema files and event files) from the data files and deploy them to QXI.

For SI API, QGen can pull the schemas from Native API framework and deploy them to QXI.

To modify an API, redeploy the API from QGen to QXI. Only custom APIs can be modified.

QGen does not work for Fin API. User has to use the QXI function to add/modify Fin APIs.



To add/modify API in QXtend, navigate to the configuration tab, select schemas node in tree menu, and then select the application version.

Click on Add button to add an API, or select a custom schema and click on Modify button to modify it.

Here are the steps for adding an API:

A screen displays stating that QXtend is active. Because changing a live running QXtend system is potentially dangerous to your system data, you are given three options to allow you to proceed safely:

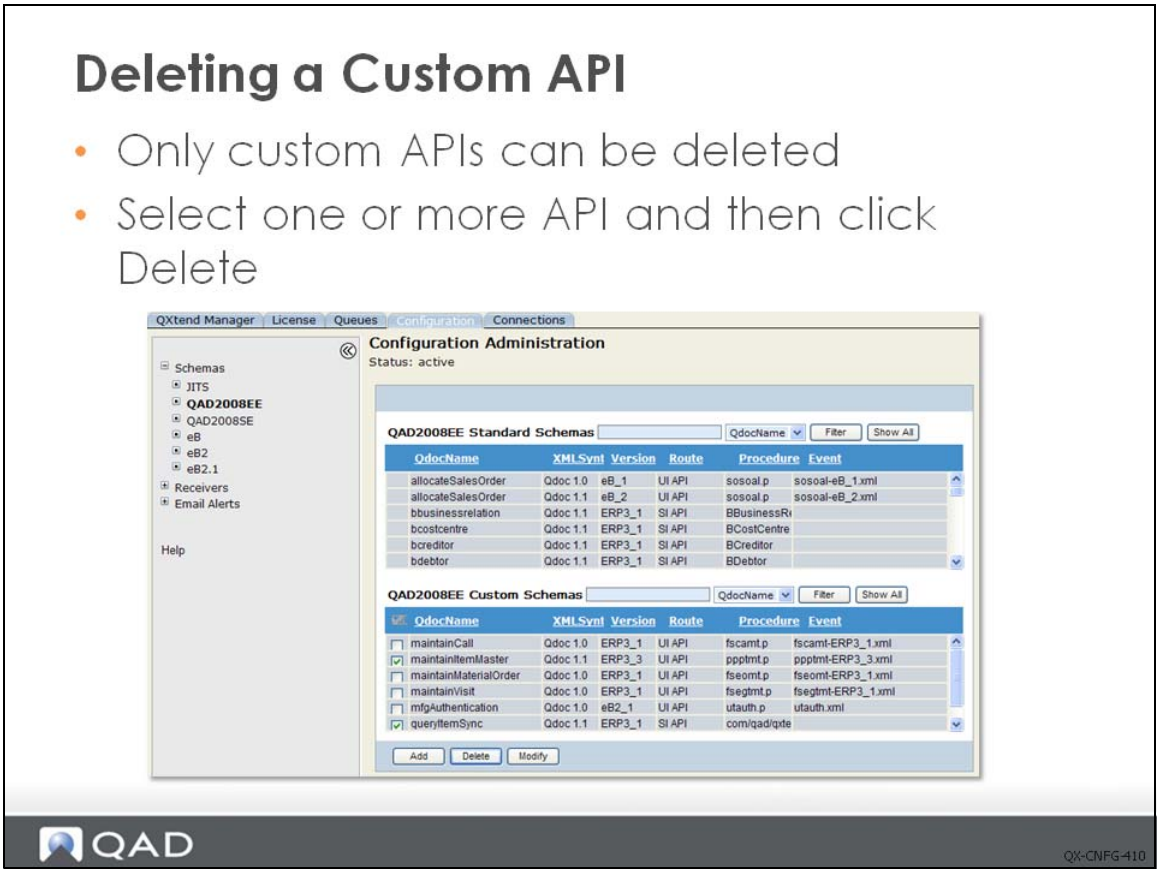
- Cancel Update: Stops the requested update. No changes are applied.
- Suspend QXtend: Suspends QXI while the update is made. During suspension, QXI cannot process requests.
- Continue without Suspending: Allows you to continue and make the configuration changes without suspending QXtend. Some changes made will not be applied until QXtend has been restarted, as many of the configuration details are cached at system startup.

After selecting either the Suspend QXtend or Continue without Suspending option, you are prompted to enter details for the API you want to add. Currently you can add APIs for UI API, Fin API or SI API adapter. Each adapter requires different parameters.

Once the details required for the new API have been entered, click Next. You can then select the receivers to automatically add this API to. If you do not want to automatically add it, do not select any receivers from the list, and just click Done. If you choose not to add the API to any receivers, you can add the API to receivers later.

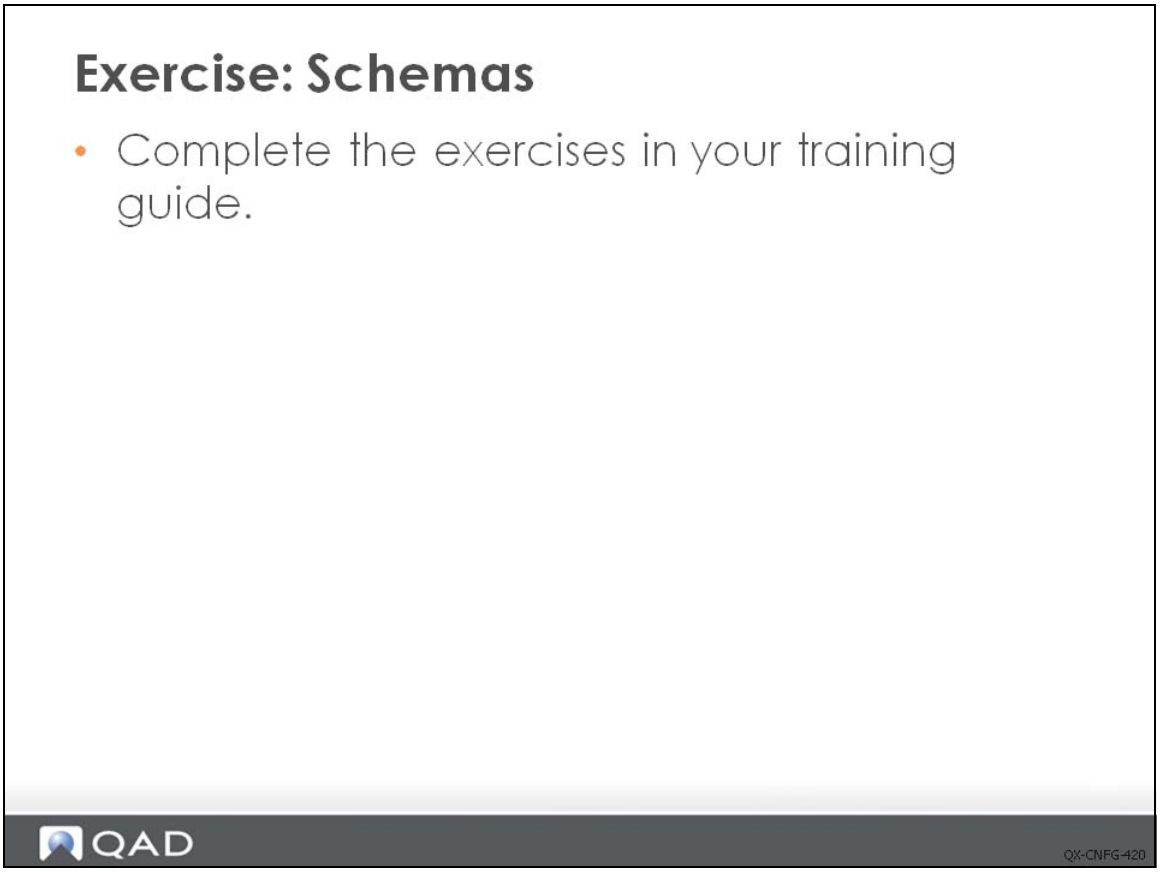
If you selected receivers, the system displays a message confirming the receivers to which the API was added.

Deleting a Custom API



Custom APIs can be deleted. To delete an API, select an application version from the Schemas node in the tree menu on the Configuration tab. Select the custom API you want to delete from the list of APIs and click Modify. You can delete more than one API at a time.

Exercise: Schemas



The following list shows a number of key concepts used in schemas in QXI. In each statement below, fill in the correct term from the list.

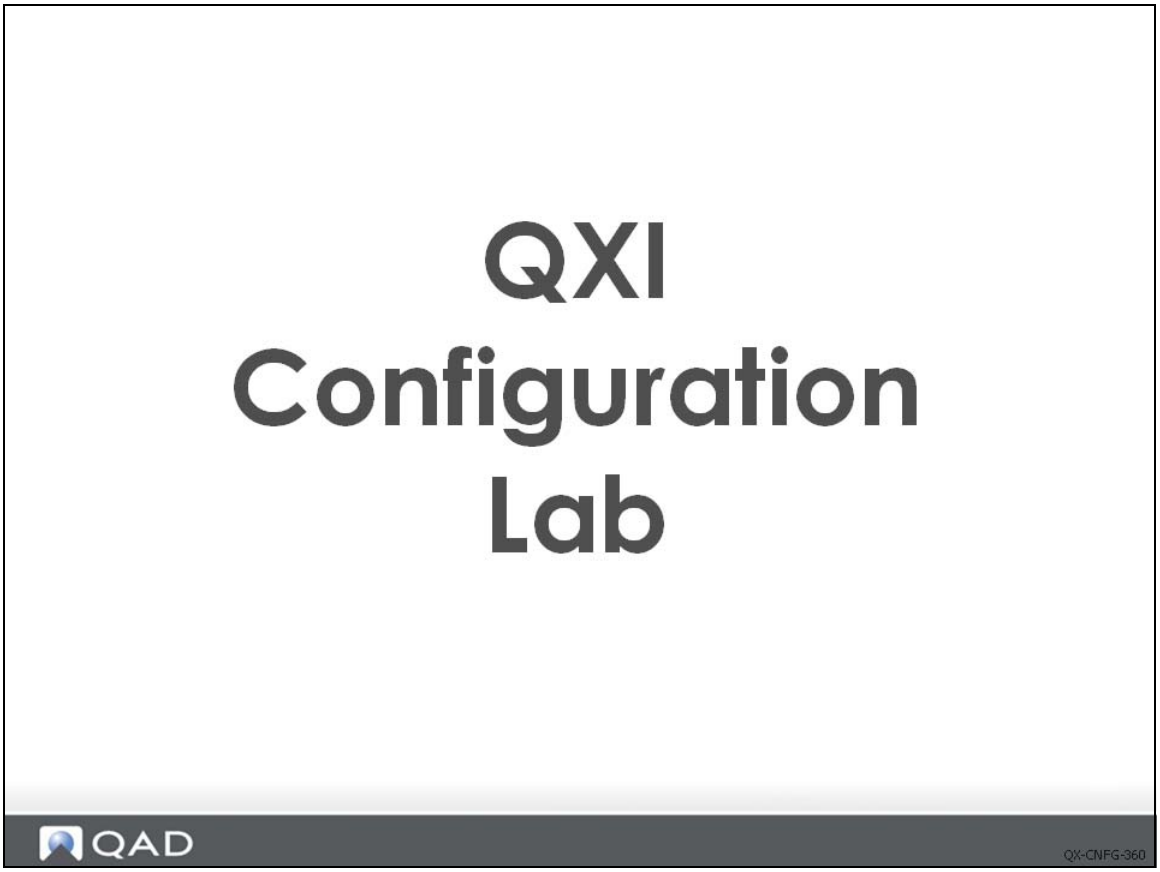
apiName	application type
response message	QGen
receivers	standard
custom	modified or deleted
version	request message
suspend	apiVersion
Schemas	

1. An API has two schemas; one schema defines the content of the _____, the other defines the content of the _____.
2. The combination of _____ and _____ combine to create a unique identifier for the API. A specific _____ of an API is linked directly to an adapter.
3. When APIs are added to QXtend, they are added for a specific _____ and can only be used on _____ that are linked to the same application type.
4. APIs in QXI can be grouped into two types: _____ and _____.
5. Standard APIs cannot be _____ since they are controlled by QAD.

6 You are given the option to _____ the QAD QXtend system when adding a new API.

7 There are two ways to add APIs to QXI, deploy APIs through _____, or add _____ in QXI.

Lab: QXI Configuration



The QAD QXtend Fundamentals Training environment has been created exclusively for use with this QAD QXtend course. This environment is available through the Training Success site http://training.success.qad.com. Each student has been given a unique user ID and password to access the training image.

QAD QXtend 1.7.2 has been installed. Both QXI and QXO are available but neither is configured. In this following lab exercises, you will configure QXI and process some test transactions to validate the QXI installation and configuration.

`<LabHomeDirectory> = C:\QXtendTraining\Labs\01-QXI-Config\`

1. Access and Start the Training Environment

1 Log in to the QAD Training Success site using the user ID and password provided to you.

2 Open the QAD QXtend Fundamentals Training image and start the Windows client provided with the image.

3 Open Internet Explorer and verify that QXI and QXO are running. The user for both is `admin` and the password is `mfgpro`.

- `QXI – http://qaddemo:8080/qxi`
- `QXO – http://qaddemo:8080/qxo`

2. Create the Receiver

Setting up a receiver is the first step in configuring QXI. The receiver identifies the destination application instance, and the APIs that are available and supported. To create a receiver:

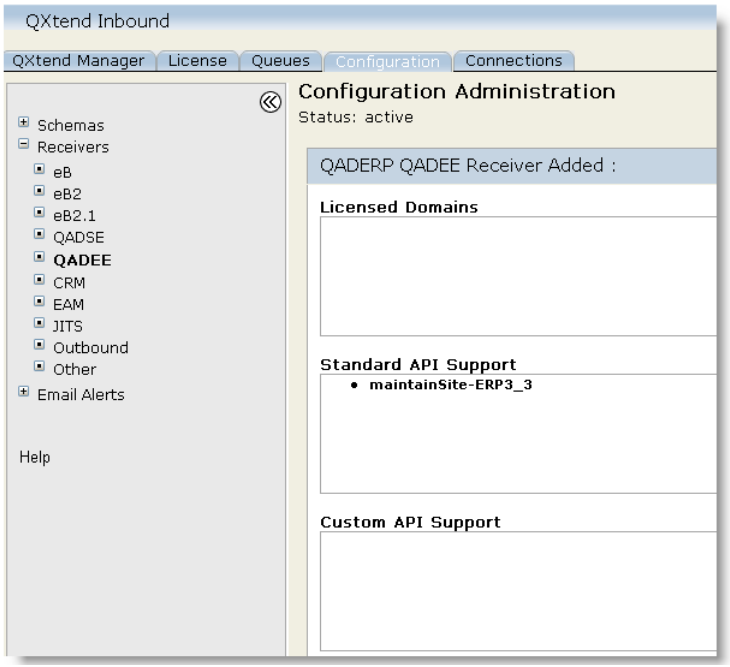
1. Open the QXI Web application in Internet Explorer:

   `http://qaddemo:8080/qxi`

2. Select the Configuration tab.
3. Select the `Receivers` node on the menu tree.
4. Select `QADEE` from the Receivers menu.
5. Click the Add button.
6. Select the Continue Configuration update without suspending QXtend Inbound option and then click Submit.
7. Set the following values:

Receiver	QADERP
Description	
Require Authentication	false
Licensed Domains	

8. Click Next to continue to the API Selection screen.
9. Enable the maintainSite API (version ERP3_3). Ensure that you select the QDoc 1.1 XML Syntax API.
10. Complete creating the receiver by clicking Done.
11. The following Receiver creation report appears.



Although the receiver has been created, it cannot be used until it can connect to the target QAD Enterprise Applications instance. You'll now define this connection.

3. Create a Connection Pool

Connection pools in QXI represent the actual connections that are available to QAD QXtend to load data into a specific instance of the QAD Enterprise Application. The connection pool is linked to a specific receiver through its name; therefore the name of the receiver and the name of the connection pool must be identical.

3.1 Create a New Connection Pool

To create a connection pool:

1. Open the QXI Web application in Internet Explorer:

   `http://qaddemo:8080/qxi`

2. Select the Connections tab.
3. Expand the Add Connection Pool node of the menu tree.
4. Select the Add UIAPI Pool option.
5. Set the following values:

Pool Name	`QADERP` (Must be identical to the receiver name created in the previous step, including letter case.)
Host	`qaddemo`
Server Startup Script	`login:|mfg|Password:|$PASSWD\|$| /dr01/qadapps/qea/qxtend/scripts/client.qxtend`
Server Startup Password	`MFGpr0`
Connection Setup User ID	`qmi`
Connection Setup Password	Blank (No password required.)
Domain	10USA (This is only the default domain. The domain can be passed with the request and will override this value when processing the request.)

6. Click the Save button to create the new connection pool.

3.2 Validate the New Connection Pool

Now that you have created the connection pool, you need to ensure the configuration is correct and that you have idle sessions available.

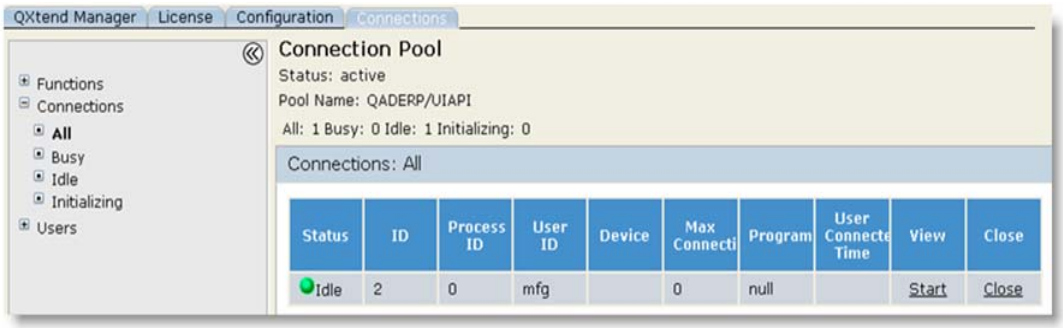
To verify that you have idle sessions:

1. Open the QXI Web application in Internet Explorer:

   `http://qaddemo:8080/qxi`

2. Select the Connections tab.
3. Expand the View Connection Pool node in the menu tree.
4. Select the `QADERP.UIAPI` option from the menu tree.

5 Expand the Connections node in the menu tree and select the All option. You should see the connection pool with a single idle connection as shown below. If there is not an idle session, the connection pool has not been configured correctly.



3.3 Verify Configuration

An idle session in a connection pool only confirms that the connection pool can log in and initialize a connection to QAD Enterprise Applications—it does not mean that requests will process successfully. It is only when requests are processed that a program is launched and you are logged into the application properly.

To verify that configuration is correct, you can use the UI Adapter Connection Test function in the Test Harness to test that a program can be started, and that data can be passed from the connection pool to the application.

Important This verification step is required only for the UIAPI adapter connection pools.

To verify your configuration:

1 Open the QXI Web application in Internet Explorer:

`http://qaddemo:8080/qxi`

2 Select the QXtend Manager tab, then click the Functions node in the menu tree.

3 Expand the Test Harness node.

4 Select the UI Adapter Connection Test node.

5 In the Receiver field, type `QADERP` and click the Submit button. A screen displays to indicate the success of the test:

4. Adding APIs to a Receiver



The receiver maintains a list of the APIs that are supported by that receiver; typically a subset of the APIs are supported by QAD QXtend. Only the APIs that are used should be enabled for each receiver. When we created the QADERP receiver earlier, we only added support for the Site Maintenance QDoc. We now also need to add support for the Item Maintenance QDoc.

1 Open the QXI Web application in Internet Explorer:

```
http://qaddemo:8080/qxi
```

2 Select the Configuration tab.

3 Select the Receivers node on the menu tree.

4 Select QADEE from the list of receivers.

5 Select the check box next to the QADERP receiver.

6 Click the Modify API button.

7 Choose the Continue Configuration update without suspending QXtend Inbound option and click the Submit button.

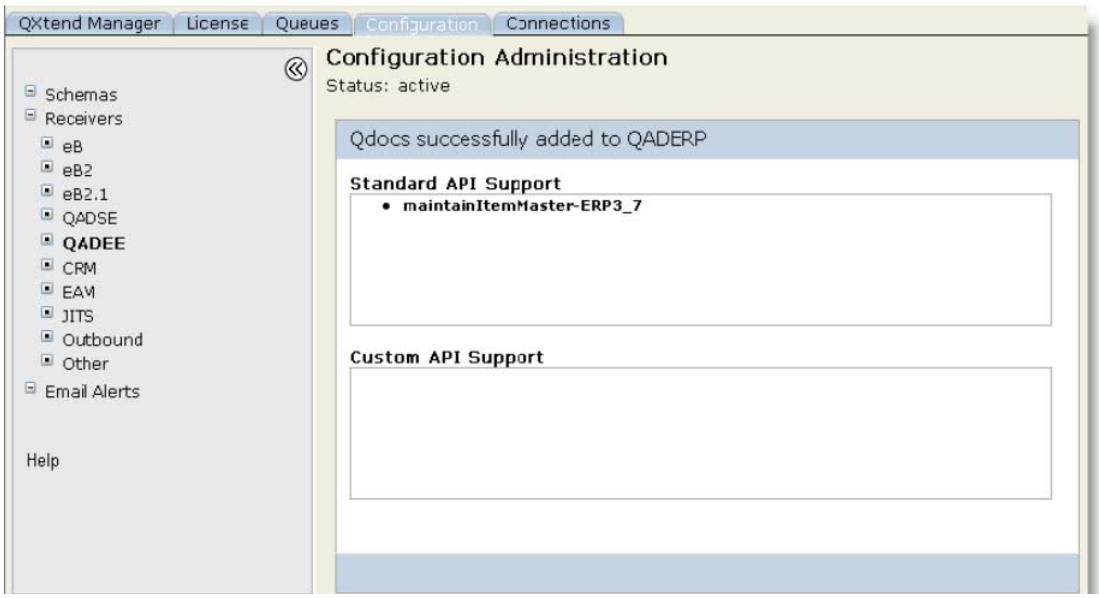
8 Click the Next button.

9 Click the Add QDoc button.

10 Locate the maintainItemMaster API and select the check box for the ERP3_7 version that uses the QDoc 1.1 XML syntax.

11 Click the Finish button. The receiver update report displays.

5. Processing a Request using the Test Harness



So far we have created the configuration required to process QDoc requests with QAD QXtend—we have not processed any messages. After configuring the receiver, creating the connection pool, and validating the configuration, we are ready to test the APIs within QAD QXtend.

The QDoc SOAP messages that are processed by QAD QXtend have a specific, required format. The content of the API message will, of course, vary by API, so how do you know what data is available for an API? QAD QXtend Inbound has several useful features to help you test your configuration, which are part of the Test Harness.

This exercise describes how to generate a sample message, edit it, and process it using the Test Harness.

5.1 Generate Sample Request QDoc

The QADERP receiver that you created earlier supports two APIs: maintainSite and maintainItemMaster. You need to generate a sample QDoc for the maintainSite API.

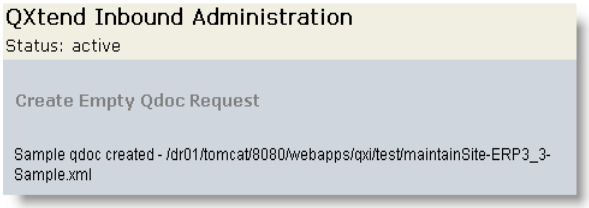
1. Open the QXI Web application in Internet Explorer:

   `http://qaddemo:8080/qxi`

2. Select the QXtend Manager tab.
3. Select the Functions node on the menu tree.
4. Expand the Test Harness node.
5. Select the Create Empty QDoc function.
6. Enter the following details:

QDoc Name	maintainSite
QDoc Version	ERP3_3
Receiver	QADERP

7. Click the Submit button. The following confirmation message displays.



The sample QDoc has been created on the server that is running QXI, and the path shown above indicates where the sample request was created on the server.

5.2 Edit the Sample Request

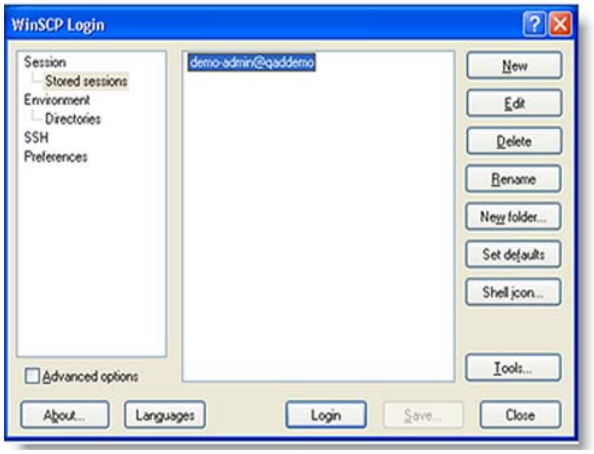
Editing XML on the Linux server that is running QAD QXtend is possible, but because it is so difficult, doing so often results in invalid XML documents that cannot be processed by QAD QXtend. Instead you should copy the file from the server to a Windows client where the XML can be edited with a free XML editor such as Liquid XML. This freeware application has been loaded onto the image being used for this training.

To transfer the file:

1 Open the Win SCP application that is installed on the Windows image.



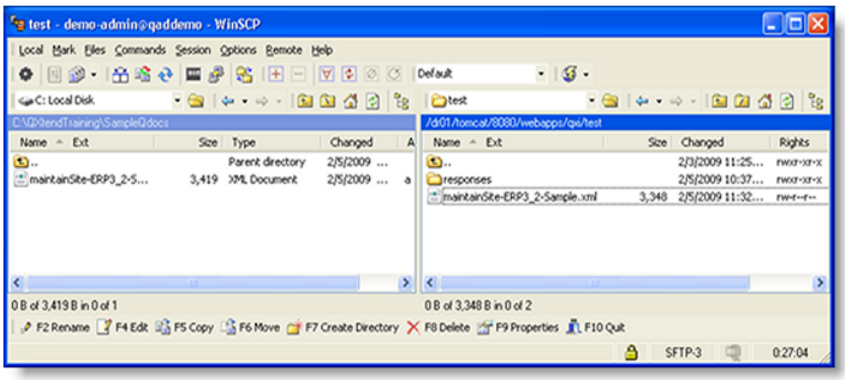
2 Click the Login button to open the stored session `demo-admin@qaddemo`.



3 Navigate to the location of the generated sample document:

`/dr01/tomcat/8080/webapps/qxi/test`

4 Copy the sample file to *<LabHomeDirectory>*`\SampleQdocs`.



5 In Notepad open the request copied to *`C:\QXtendTraining`*`\SampleQdocs`.

6 Review the contents of the generated QDoc message. Note that the area that contains the application data does not contain valid data and the data must be edited.

The `wsa:To node` contains the receiver ID that will process the request. This is one of the most important pieces of information as this controls which application instance the message is loaded into.

7 Make the following changes to the XML:

a Change the `wsa:Address` node to:

`<wsa:Address>urn:services-qad-com:qxtendtrain</wsa:Address>`

This node contains the sender ID and is required because the QAD QXtend instance we are using at the moment is only available for data synchronization. By setting the sender ID to `qxtendtrain`, any message with that ID can be processed without raising a licensing exception. The `qxtendtrain` sender ID is only valid for this image and will not work on other QAD QXtend installations.

b Set the `propertyValue` element for the domain `ttContext` parameter to `10USA`.

c Delete all the nodes for the site iteration except the `siSite`, `siDesc` and `siStatus` nodes, and set the following values:

siSite	QMI PR
siDesc	QMI Puerto Rico (New)
siStatus	N-N-N

You can use Internet Explore to view the XML file. Your request should now look like this:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <soapenv:Envelope xmlns="urn:schemas-qad-com:xml-services" xmlns:qcom="urn:schemas-qad-
  com:xml-services:common" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsa="http://www.w3.org/2005/08/addressing">
 - <soapenv:Header>
    <wsa:Action />
    <wsa:To>urn:services-qad-com:QADERP</wsa:To>
    <wsa:MessageID>urn:services-qad-com::QADERP</wsa:MessageID>
   - <wsa:ReferenceParameters>
      <qcom:suppressResponseDetail>true</qcom:suppressResponseDetail>
    </wsa:ReferenceParameters>
   + <wsa:ReplyTo>
   </soapenv:Header>
 - <soapenv:Body>
   - <maintainSite>
     - <qcom:dsSessionContext>
       - <qcom:ttContext>
          <qcom:propertyQualifier>QAD</qcom:propertyQualifier>
          <qcom:propertyName>domain</qcom:propertyName>
          <qcom:propertyValue>10USA</qcom:propertyValue>
        </qcom:ttContext>
       - <qcom:ttContext>
          <qcom:propertyQualifier>QAD</qcom:propertyQualifier>
          <qcom:propertyName>version</qcom:propertyName>
          <qcom:propertyValue>ERP3_3</qcom:propertyValue>
        </qcom:ttContext>
      </qcom:dsSessionContext>
     - <dsSite>
       - <site>
          <siSite>QMI-PR</siSite>
          <siDesc>QMI Puerto Rico</siDesc>
          <siStatus>N-N-N</siStatus>
        </site>
      </dsSite>
    </maintainSite>
  </soapenv:Body>
</soapenv:Envelope>
```

d Save the file as *`C:\QXtendTraining`* `\SampleQdocs\SiteReq01.xml`.

e Copy the file just saved onto the Linux machine using WinSCP. The file needs to be copied to:

`/dr01/tomcat/8080/webapps/qxi/WEB-INF/receivers/QADERP/requests`

This directory is used by the Test Harness to retrieve the test requests.

5.3 Verify QAD Data

Before running the test message through QAD QXtend, log into QAD Enterprise Edition and confirm the description for the QMI PR site you are updating.

5.4 Process Request

Now we process the request using the Process Request option on the Test Harness menu.

1. Open the QXI Web application in Internet Explorer:

   `http://qaddemo:8080/qxi`

2. Select the QXtend Manager tab.
3. Expand the Test Harness node on the menu tree.
4. Select the Process Request function.
5. Enter the following values:



6. Click the Submit button. QAD QXtend processes the response and displays the success status when the response is received.

5.5 Process Request with SOAP UI

The Test Harness provides a useful set of tools for processing requests and testing our APIs. However, generating requests, editing them, and moving them around is time-consuming and prone to error. The soapUI product is a free toolset that provides a testing framework for Web services that follow the WS-I specifications. This tool can be downloaded from `www.soapui.org`. Two versions are available: soapUI, which is a free edition, and soapUI Pro, which provides additional features.

Using soapUI is a fast way to build test SOAP messages using the API WSDL and schemas shipped with the QAD QXtend APIs. It also helps you build a suite of test cases for any API that is supported by QXI.

5.5.1 Use Predefined soapUI Project

The following exercise shows you how to use soapUI to test QXI using a predefined project and set of requests:

1 Open soapUI on the Windows image, use the shortcut on the Desktop.



2 Open the Lab 01 – QXI Configuration workspace.

   a File – Switch Workspace.

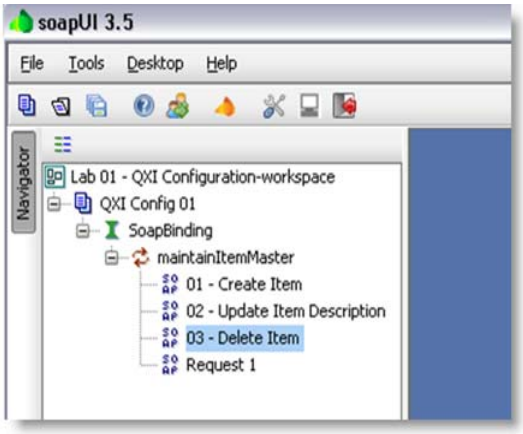
   b Browse to *<LabHomeDirectory>*

   c Select the `Lab 01 - QXI Configuration-workspace.xml` file.

3 Expand the QXI Config 01 project.

4 Expand the SoapBinding level.

5 Expand the maintainItemMaster level.



6 Open the Request 1 message and review the structure of the SOAP message. None of the parameters are prepopulated with data, and no default session context records are created, but the basic structure of the message has been created from the WSDL and the API request schema. The following messages have been created from this raw message and updated with the necessary values for the message to process.

7 Open the 01 – Create Item message and review the contents of the message:

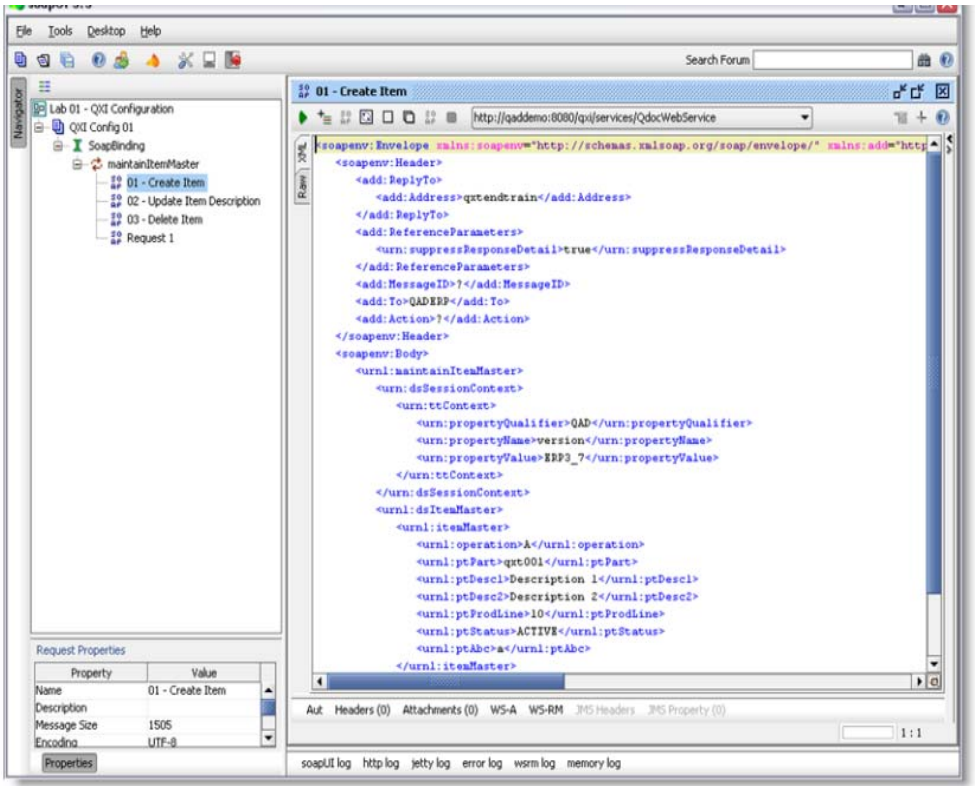
   a Locate and validate the receiver ID.

   b Locate and validate the sender ID.

   c Validate the session context entries.

   d Validate the application data section.

   e Process the request by clicking the green arrow button in the request window.



f Validate that the item has been created in the application with the values specified in the request.

8 Open the 02 – Update Item Description message and review the contents of the message:

a Repeat the step from the previous message.

b This QDoc updates the item created in the previous message.

c Validate the update is successful.

9 Open the 03 –Delete Item message and review the contents of the message:

a Repeat the step from the previous message.

b This QDoc deletes the item updated in the previous message.

c Validate the delete is successful.

5.5.2 Generate WSDL

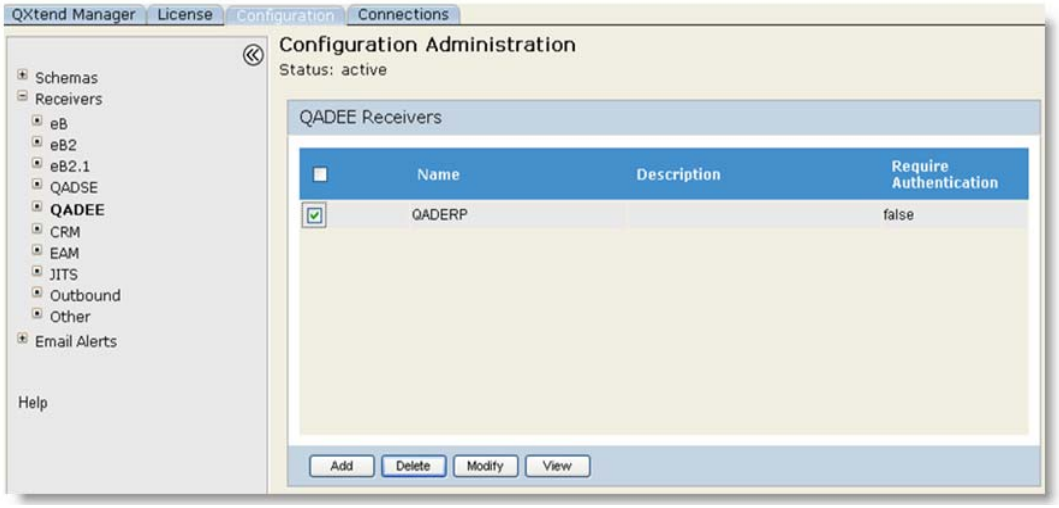
1 Open the QXI Web application in Internet Explorer:
`http://qaddemo:8080/qxi`

2 Select the Configuration tab.

3 Select the Receivers node on the menu tree.

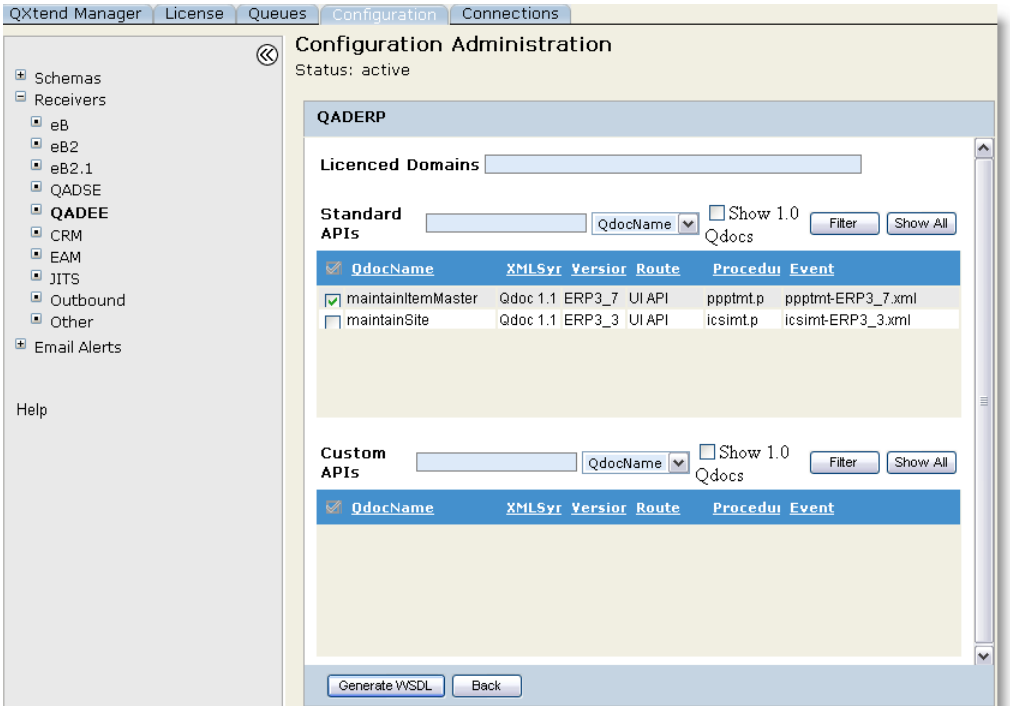
4 Select QADEE from the list of receivers.

5 Select the check box next to the QADERP receiver.



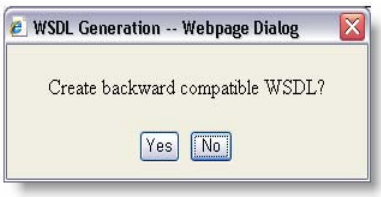
6 Click the View button.

7 Select maintainItemMaster from the list of Standard APIS.

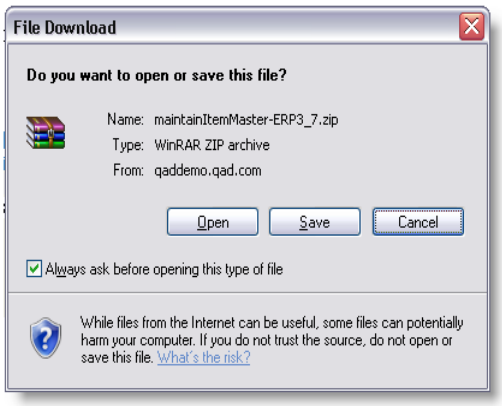


8 Click the Generate WSDL button.

9 Click the No button for “Create backward compatible WSDL?”.



10 Save the maintainItemMaster-ERP3_7.zip to *<LabHomeDirectory>*.



11 Unzip the file to *<LabHomeDirectory>*\maintainItemMaster-ERP3_7.

5.5.3 Create a New soapUI Project

From within the Lab 01 – QXI Configuration workspace, do the following to build a new project.

1 Create a new project:
- File – New soapUI Project.

2 Project Name = QXI Config 02.

3 Browse to this WSDL file:

<LabHomeDirectory>\maintainItemMaster-ERP3_7\maintainItemMaster-ERP3_7.wsdl

4 Ensure that the Create Requests check box is selected.

5 Click OK to create the new project.

6 Drill down to the Request 1 message created under the QXI Config 02 project.

7 Right-click Request 1 and select the Clone Request option. Enter a name for the new request.

8 Edit the new request so that it can be processed by QXI. Change the SOAP header:

a Specify the receiver as `QADERP`.

b Specify the sender to be `qxtendtrain`.

c Set `suppressResponseDetail` to `true`.

9 Create the necessary session context entries:

   a Create two ttContext iterations.

   b Qualifier = `QAD`, Name = `version`, Value = `ERP3_7`

   c Qualifier = `QAD`, Name = `domain`, Value = `10USA`

Note If you copy and paste the empty `ttContext` node and blank lines are created, right-click the request and select the Format XML option. soapUI will remove any blank lines and correct the indentation of the XML.

10 Edit the application data section of the message as follows:

   a The `dsItemMaster` node by default will contain all of the fields available when processing an Item Master update. However, only a few fields are required for a transaction to process successfully. The best way to identify the mandatory fields is to run through the UI and see which fields you have to enter data for.

   b Item Master in this instance of QAD EE only has a few mandatory fields. The list of fields may change depending on configuration:

   - `ptPart`
   - `ptProdLine`
   - `ptStatus`
   - `ptAbc`
   - Recommended fields are `ptUm`, `ptDesc1`, `ptDesc2`, `ptLoc`, `ptShipWtUm` and `ptNetWtUm`.

   c Edit the XML and create a new item QDoc that will be processed successfully by QXI. If you do not want to use a node, delete it from the XML.

You are now ready to process the request that you have created in the SOAP UI project.

11 Process the QDoc message that you created in the new soapUI project and verify that the Item is created successfully.

12 Create and process a new message and update the Item created in the previous step.

13 Verify the changes in QAD EE.

14 Create and process a new message that will delete the item updated in the previous step.

15 Verify the changes in QAD EE.

6. Load New API

New APIs are developed during the implementation phase of QAD QXtend. It is important to understand how these APIs are deployed to QAD QXtend. New APIs could be developed with QGen if it uses the UI API adapter, or a native OpenEdge API that uses the Service Interface adapter. Regardless of the adapter type, each API needs to have a request schema and a response schema.

QGen provides the function to deploy an API to QXtend Inbound directly. Later we will practise that in the lab for QGen. In this lab, we use another approach to load API to QAD QXtend.

The API that you will deploy to QAD QXtend is for the Customer Item Maintenance function and leverages the UI API adapter. This new version needs to be deployed to QADEE. The schema and event files for the API have been provided.

To deploy an API:

1 Open the QXI Web application in Internet Explorer:

`http://qaddemo:8080/qxi`

2 Select the Configuration tab.

3 Select the Schemas node in the menu tree.

4 Select QADEE.

5 Click the Add button.

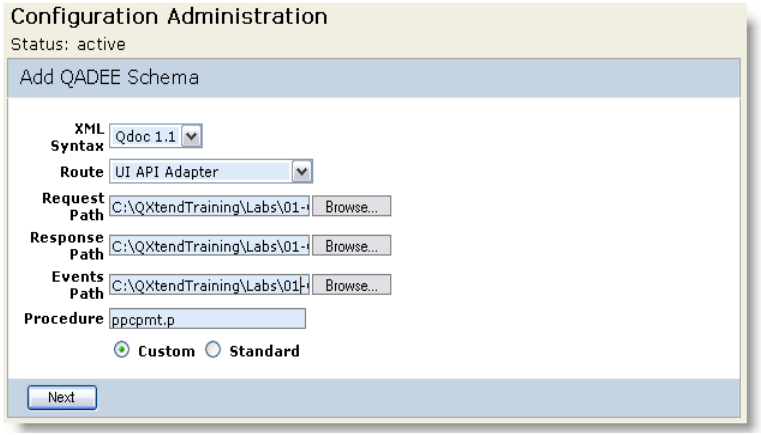
6 Choose the Continue Configuration update without suspending QXtend Inbound option and click the Submit button.

7 Set the following values:

XML Syntax	1.1
Route	UI API Adapter
Request Path	*<LabHomeDirectory>*`\Schemas\maintainCustomerItem-ERP3_1.xsd`
Response Path	*<LabHomeDirectory>*`\Schemas\maintainCustomerItemResponse-ERP3_1.xsd`
Events Path	*<LabHomeDirectory>*`\Schemas\ppcpmt-ERP3_1.xml`
Procedure	`ppcpmt.p`

8 Deploy this API as a custom API.

9 Click the Next button.



10 Add the API to the QADERP receiver by selecting the check box. Then click the Done button.

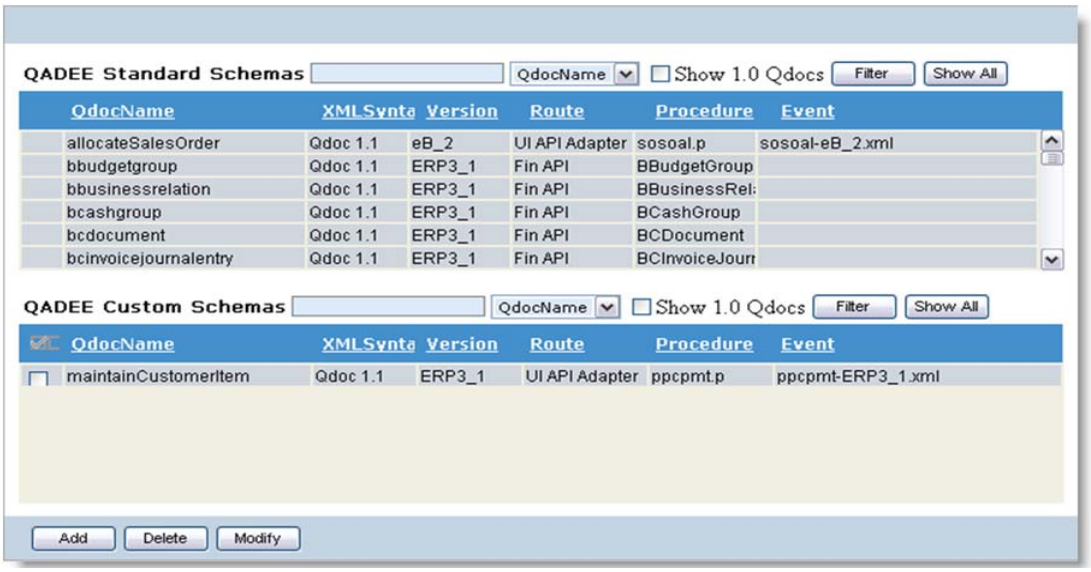
7. Verify the Customer Item API has been added

To verify that the Maintain Customer Item API was successfully added to QXI, you need to check two places. First check that the API is available as a custom API for the QADEE application type; second, check it is a supported API on the QADERP receiver.

1 In the Configuration tab, select the Schemas node on the menu tree.

2 Select QADEE.

The maintainCustomerItem version ERP3_1 should display as the only custom API that is available for the QADEE application type as shown here:

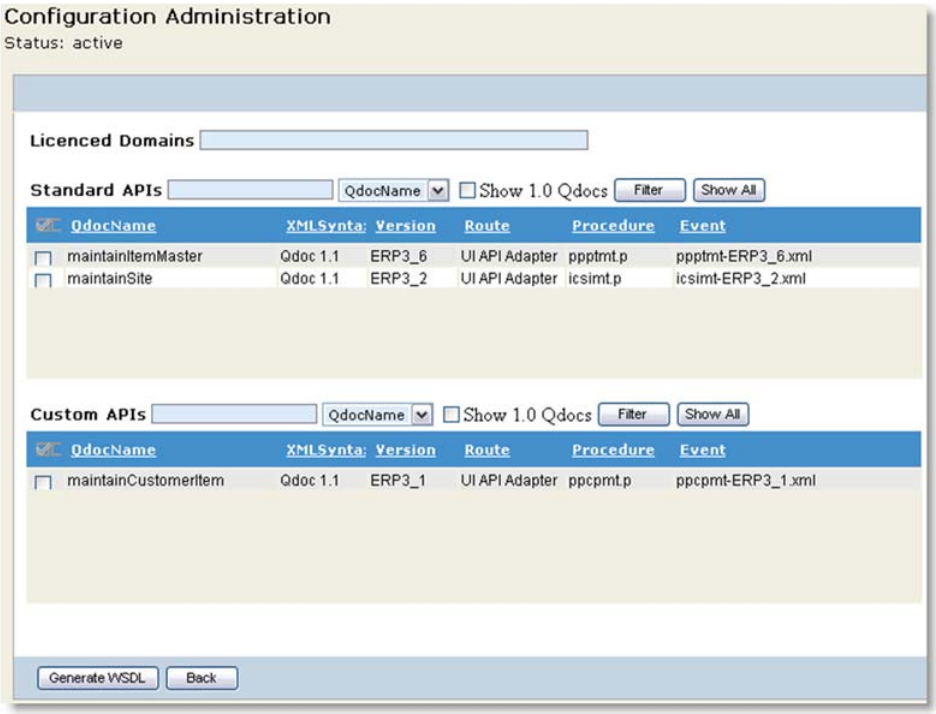


3 Select the Receivers node on the menu tree.

4 Select QADEE.

5 Select the check box next to the QADERP receiver and click the View button.

The maintainCustomerItem API version ERP3_1 should display as a supported QDoc for the receiver as shown here:



8. Test the New Customer Item Maintenance API

The new version of the Customer Item Maintenance API has now been successfully deployed to QXI. However, until the new version has been used to process a couple of transactions, you cannot be sure that everything has been uploaded correctly.

To test a QAD QXtend API you should use soapUI to build a WSDL project, create some requests, and process those requests using soapUI.

Generate the WSDL file first:

1. View the configuration of the receiver QADERP.
2. Select maintainCustomerItem from the list of Custom APIs.
3. Click the Generate WSDL button.
4. Click the No button for "Create backward compatible WSDL?"
5. Save maintainCustomerItem-ERP3_1.zip to `<LabHomeDirectory>`.
6. Unzip the file to `<LabHomeDirectory>/maintainCustomerItem-ERP3_1`.
7. Open soapUI.
8. Create a new workspace:
   a. Choose File|New Workspace.
   b. Set the workspace name to Lab 02 - QXI Schemas.
   c. Save the workspace file in `<LabHomeDirectory>`.

9 Create a new WSDL project.

a Right-click the workspace name.

b Select New soapUI Project.

c Set the project name to QXI Schema 02.

d Set the WSDL file to:

`<LabHomeDirectory>/maintainCustomerItem-ERP3_1/maintainCustomerItem-ERP3_1.wsdl`.

e Click OK to create the project.

10 Drill down to the Request 1 message created under the QXI Schema 02 project.

11 Right-click Request 1 and select the Clone Request option. Enter a name for the new request.

12 Edit the new request so that it can be processed by QXI.

13 Change the SOAP header:

a Set the receiver to `QADERP`.

b Set the sender to `qxtendtrain`.

c Set `suppressResponseDetail` to true.

14 Create the necessary session context entries:

a Create two `ttContext` iterations.

b Qualifier = `QAD`, Name = `version`, Value = `ERP3_1`.

c Qualifier = `QAD`, Name = `domain`, Value = `10USA`.

Note If you copy and paste the empty ttContext node and blank lines are created, right-click the request and select the Format XML option. soapUI will remove any blank lines and correct the indentation of the XML.

15 Edit the application data section of the message:

a The dsCustomerItem node by default will contain all the fields available when processing a Customer Item Master update. However, only a few fields are required for a transaction to process successfully. To identify the mandatory fields, run through the UI and see which fields you have to enter data for.

b Edit the XML and create a new Customer item QDoc that will be processed successfully by QXI, use the following values:

cpCust	10C1000
cpCustPart	Roxy-2134-Y
cpPart	01010
cpComment	QXtend Training Customer Item
cpCustPartd	Roxy Component
cpCustEco	35GY-001

16 Process the QDoc message that you have created in the new soapUI project and verify that the item is created successfully in Customer Item Maintenance (1.16).

17 Create and process a new message, and update the Customer Item created in the previous step so that the Display Customer Item (cpCustPartd) is Roxy Component 2134-Y.

18 Verify the changes in QAD EE.

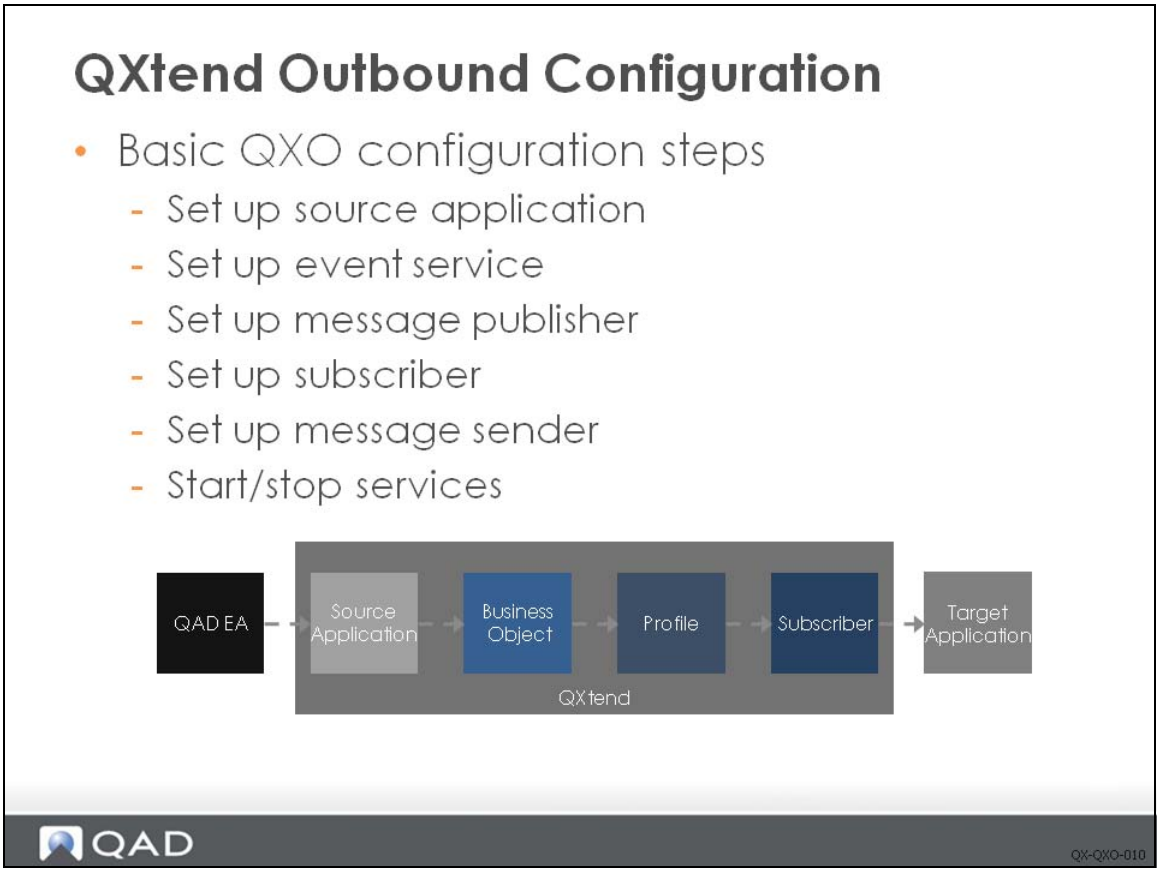
19 Create and process a new message that will delete the Customer Item updated in the previous step.

20 Verify the changes in QAD EE.

Chapter 3

QXO Configuration

Introduction



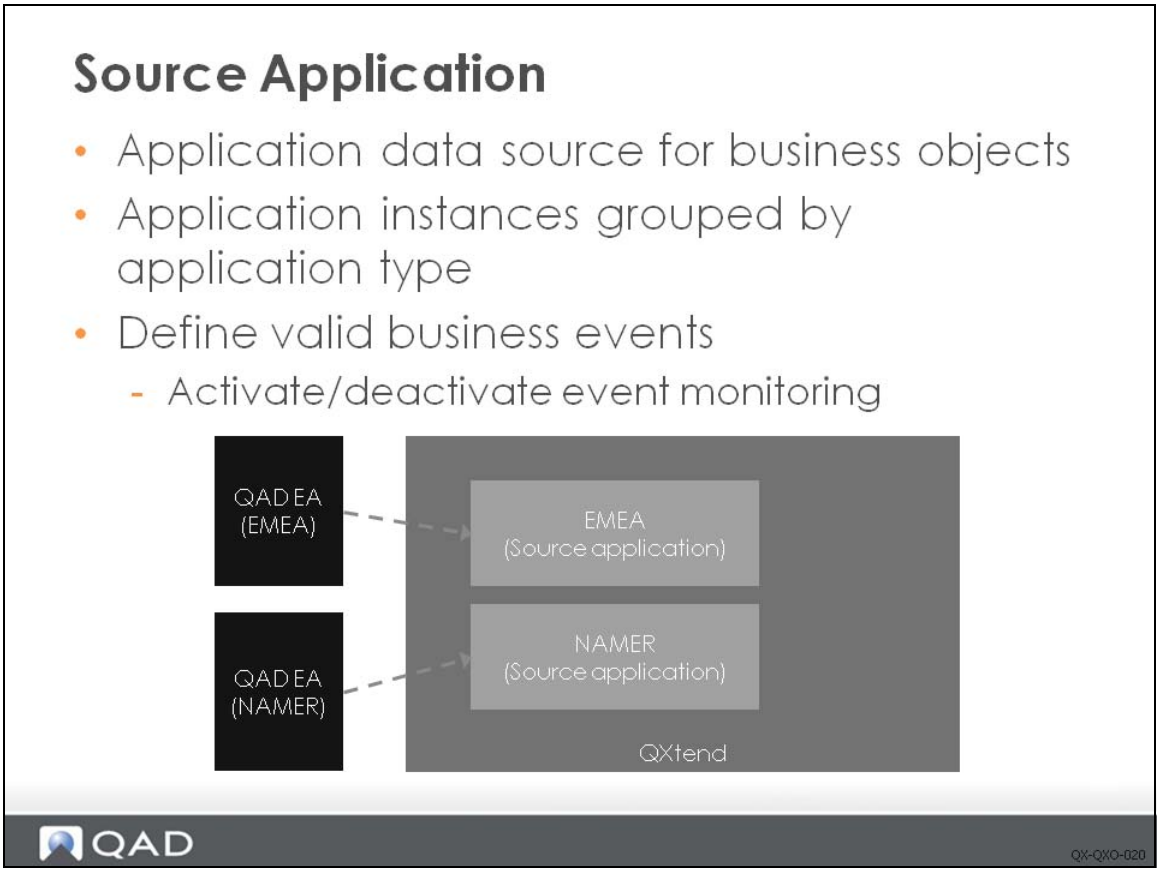
This chapter describes each component of QXO and their configuration and management.

Several steps must be followed to configure QXO to ensure that events raised by the source application are selected and published to interested applications. In addition, the services and components that drive QXO need to be created and running. The following tasks must be performed:

1. Set up the source application. The source applications provide details of the data source for messages, including the events that are active and how to connect to the source application databases.
2. Set up an event service. The event service extracts business object data from the source application after an application event has occurred.
3. Set up the message publisher. The message publisher is responsible for converting the business object message into the format that is required by the profiles defined for the business object.
4. Set up the message sender. The sender delivers the profile message to the subscribing application.
5. Start/stop services. The event service, message publisher, and the message sender are crucial services that must be running for QXO to function correctly. These services can be started and stopped manually in the QXO application.

This chapter describes these tasks.

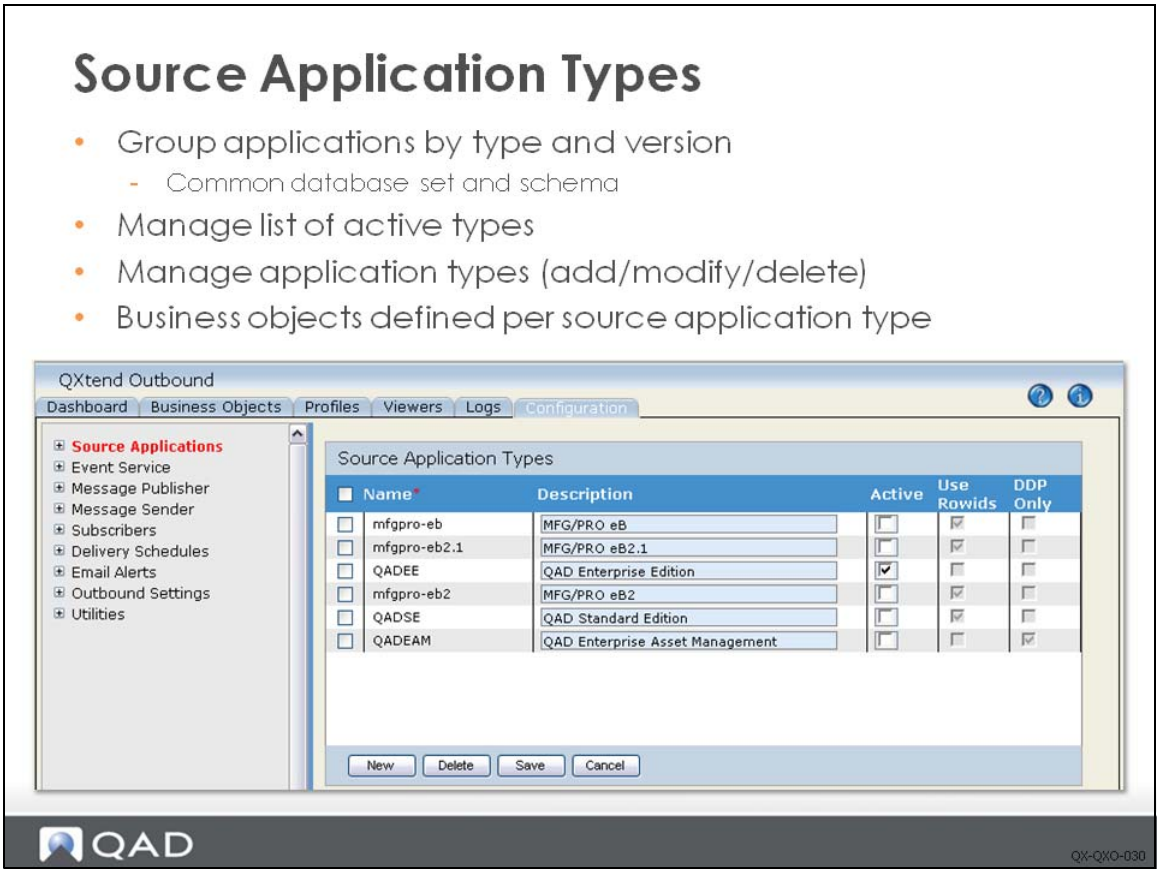
Source Applications



Source applications supply the business events and business data that drive QXO. Business events are raised by the source application and identify when changes have been made. The business event also identifies which instance of the business object has been changed. This data identifies the data that needs to be extracted by the event service. The event is used to locate the business objects to extract. The source application is configured to connect to the source application and extract the data.

The source application is assigned a unique name that identifies the instance. The source application also defines a set of databases that collectively contain all of the data for that application. The configuration contains all the information required to connect to the application database and extract data when processing an event. The source application contains a list of all valid events for that application, including which of those events types are active. Changing an event type to inactive in the source application will prevent QXO from processing any events of that type that are raised.

Source Application Types

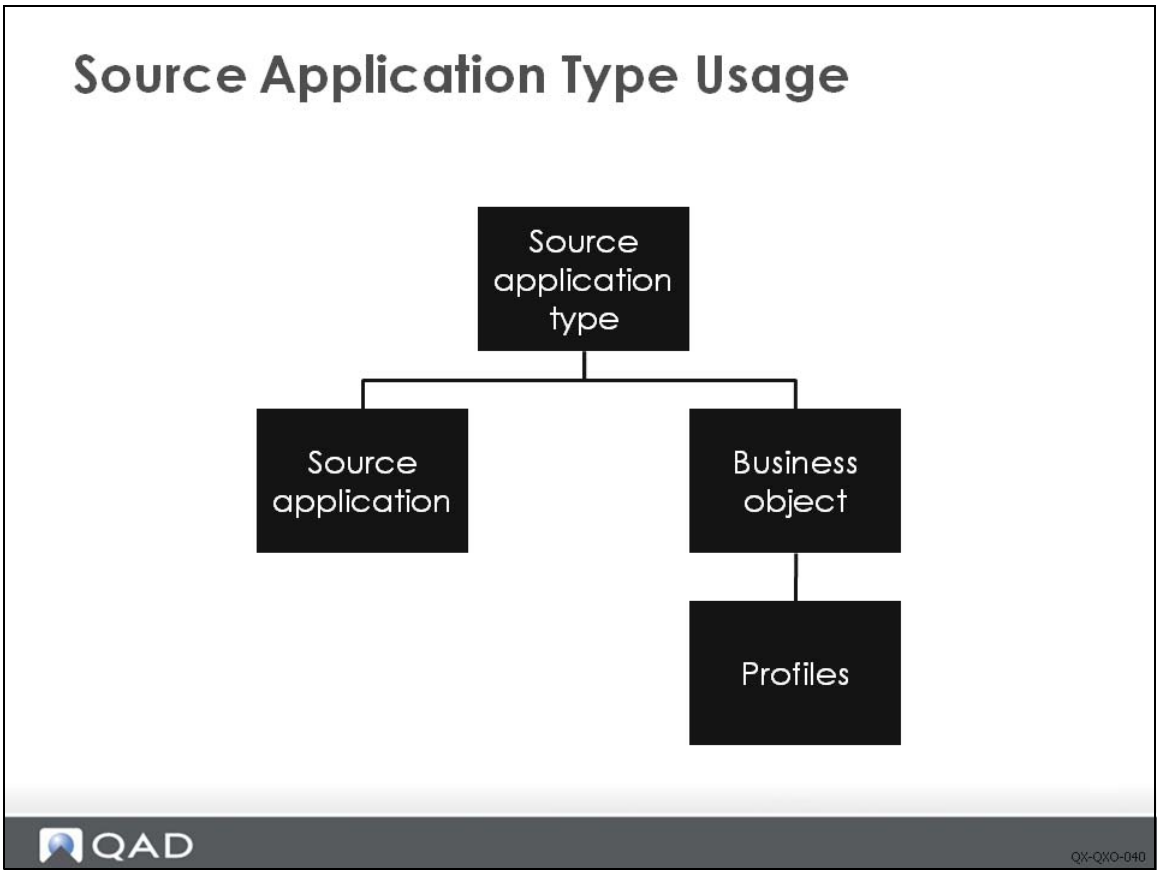


Source application instances are linked to a specific source application type. The type enables multiple instances of a QAD application release to be created. Instances of a source application type share the same set of application databases; those databases must have identical schema definitions.

Source applications can only be created for active source application types. To view the active source applications types, select the QXO Configuration tab and then select the Source Applications node in the tree menu. You can then modify the Active attribute for each type. The Active check box also controls the list of source application types displayed in the Business Objects and Profiles maintenance screens—only active source application types display.

QAD QXtend ships with a predefined list of source application types and associated business objects and profiles. You can create your own types and group your application instances under your specific type. The source applications section of QXO lets you add, modify, or delete types.

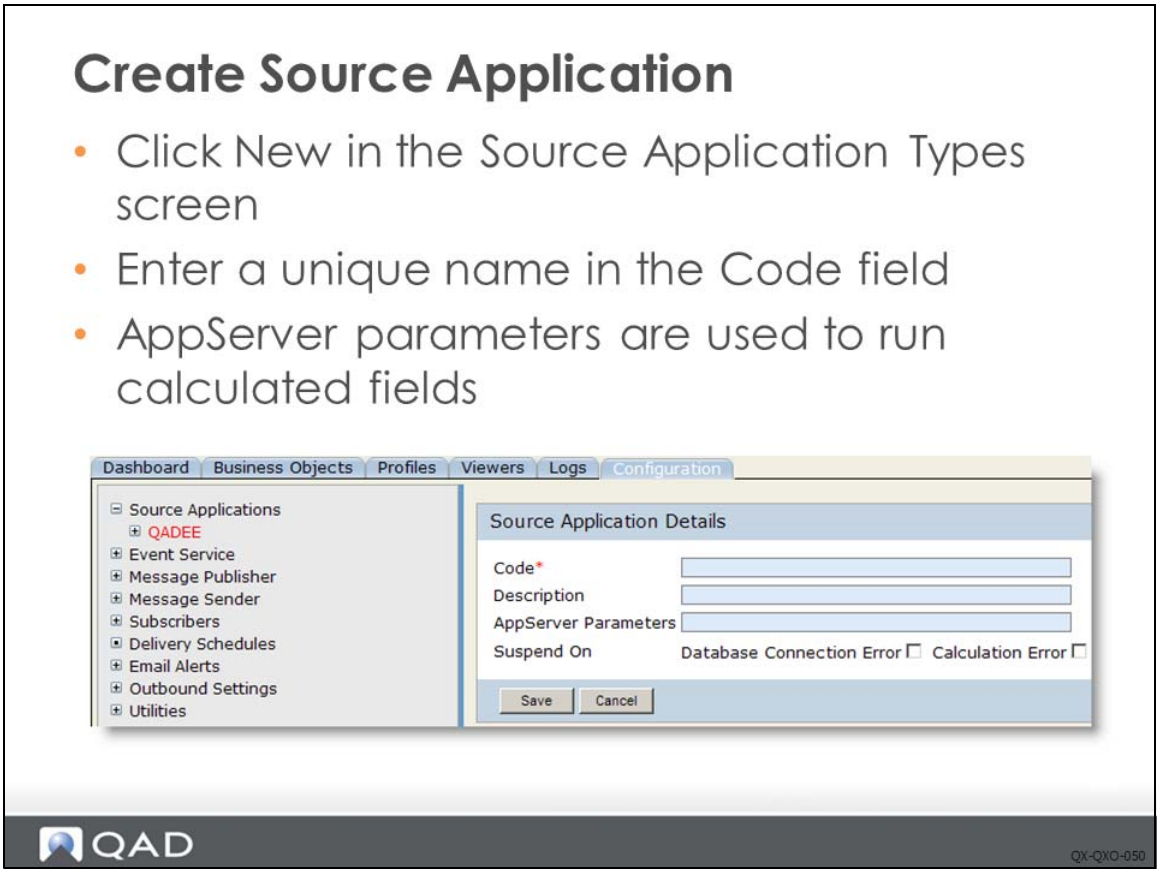
The Use Row IDs flag that is associated to a type determines how QXtend identifies the exact row of data in the source application that was changed as part of the business event that was sent to QXO. When the Use Row IDs check box is selected, the Progress rowid of the table that was changed is used to identify the changed row. When the Use Row IDs check box is not selected, the source application is responsible for publishing a unique ID for every event. If Progress rowids are used, you must be careful when performing a dump and load of any source application databases as this will change the rowid of every record. When performing a dump and load, use the mass rowid synchronization utility to synchronize the rowids in QXtend with the new rowids in the source application instance. For details see *Technical Reference: QAD QXtend*.



This slide shows how the source application type is used within QXO to group source applications and business objects into common categories that are valid for specific QAD application releases.

As described earlier, the source application defines the set of database connections—all source applications of the same type share the same set of databases and schema. The schema information of the connected databases is used to build the business objects that are available to a source application. Business objects therefore must be specific to a source application type to ensure that the tables and fields used in the business object are valid for any instance of a source application type. If the business objects were not linked to the source application type, definitions could not be shared among source applications; instead, you would have to define business objects for each source application.

Creating a Source Application



To create a source application, select the Configuration tab, and then click the Source Applications node in the tree menus. Select the source application type you want to create and then click New. Complete the following fields:

Code. The name for the source application instance. Use a meaningful name that identifies the data source and describes the location and type of the source application.

Description. (Optional) Use a meaningful description that identifies the source application instance.

AppServer Parameters. (Optional) The parameters required to connect to a Progress AppServer. The AppServer can be used to execute calculated field programs that require a connection to the source application database set.

Business objects can contain fields that need to execute business logic to derive a value that is not stored directly in the database. These types of calculations that need database access are executed on a Progress AppServer. The values you enter are used to connect to the AppServer. Typically you need to enter the name of the AppServer and the name of the host the AppServer is running on. For example:

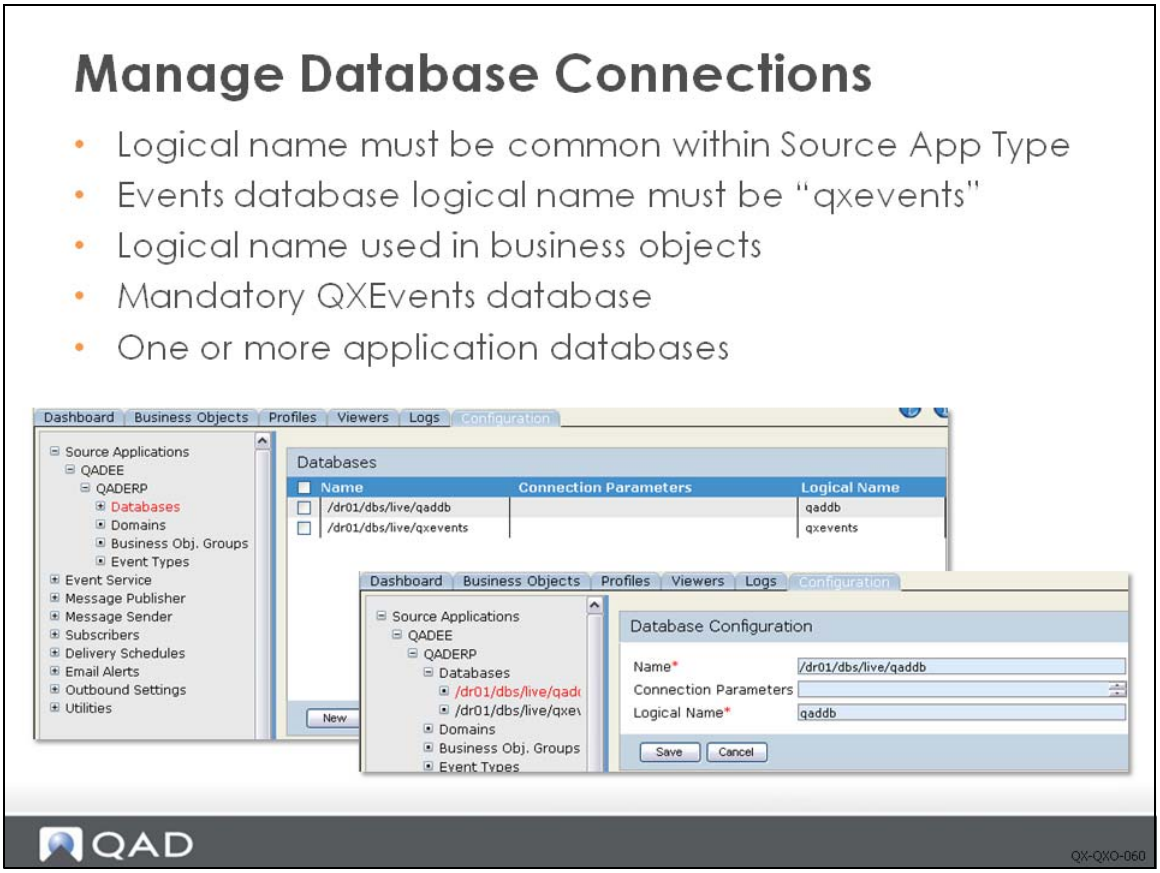
```
-AppService qxocalc_AS -h qxdemo
```

Suspend on. (Optional) Specify whether to suspend the source application on one or both of the following errors: database connection error and calculation error. All QXO services only process events and messages of active source applications and ignore events and messages of

suspended source applications. In addition, suspended source applications cannot be connected for any UI operations, such as source application configuration. For more information, refer to QXtend User Guide.

Click Save to create the new source application. You now need to configure the database connections and event types.

Managing Database Connections



Source applications connect to a set of databases that contain the source application data; every source application for a specific type must be connected to the same set of databases. Also, those databases must be connected with the same logical database name.

You can add, modify, or delete database connections. After creating a source application, navigate to the Databases menu option under the source application instance in the Source Applications node. Here you can create new database configurations.

To create or modify a database configuration, complete the following fields:

Name. The physical name of the database. When connecting with a Progress shared memory connection, this is the full path to the database. To connect using shared memory, QXO must be running on the same host as the database. Client-server connections require only the physical name of the database—no path is required. The other connection parameters are defined later. The value entered here are the same values as required by the –db Progress connection parameter.

Connection Parameters. (Optional). A string that contains the connection parameters required to connect to a Progress database in client-server mode. Typically these parameters are the host name and the service name for the database; for example:

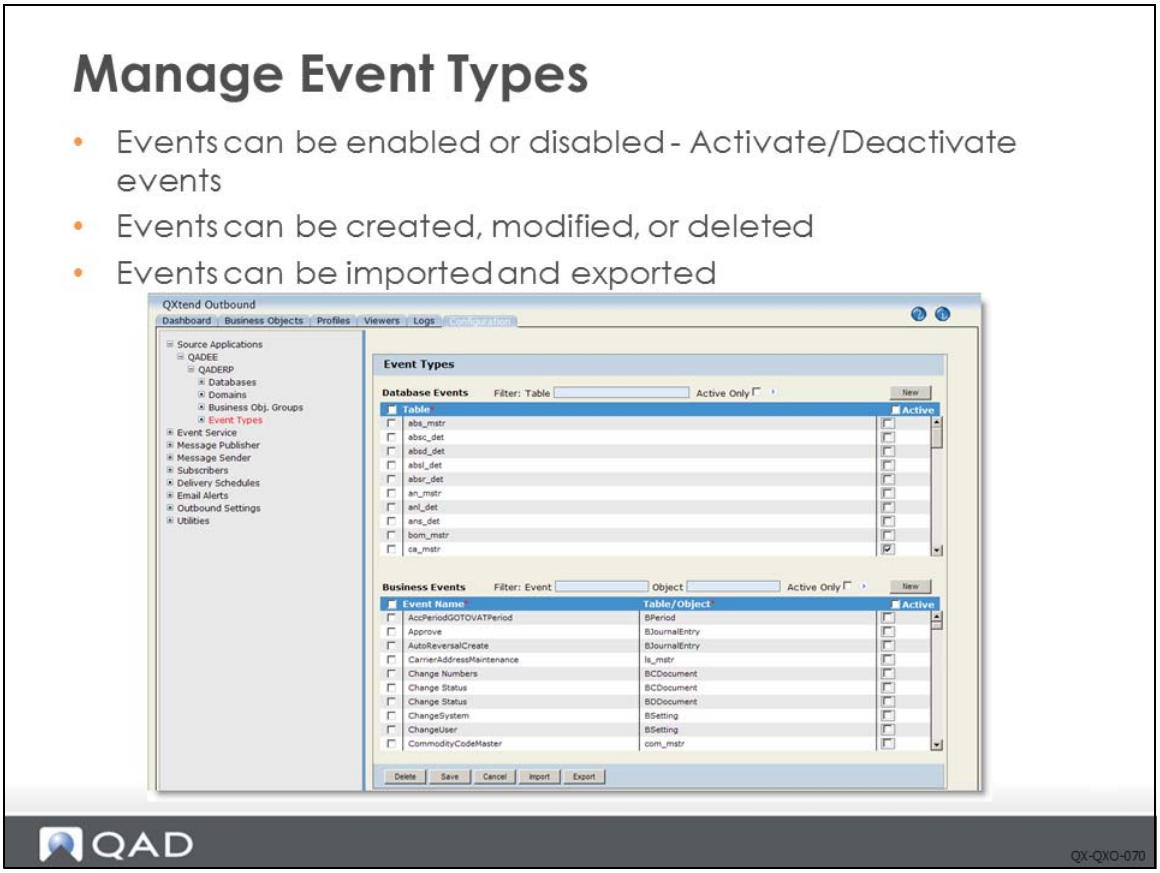
```
-H qxdemo -S mfgprod-service
```

Logical Name. The logical name associated to the database when it is connected. This name is used when populating the business object with data to ensure that the data is pulled from the correct table in the correct database. Consequently it is critical that all databases for source applications of the same source application type connect to the databases using the same logical database name.

Source applications always have a minimum of two databases defined:

- QXEvents: This database is written to by the source application and stores the details of events that have been raised and not yet processed. Events are deleted from this database once QXO has extracted the data related to the event. The logical name for every source application instance must be called `qxevents`; otherwise transaction processing in QXtend will fail.
- Application database(s): At least one application database must be configured. The application databases are the source for the business object data that QXO extracts. You can configure as many databases as required to connect to all data sources. Typically QAD implementations have additional side databases that contain information required for customer-specific customizations. In this scenario the side database can be connected to and data extracted.

Managing Event Types



QXO listens for events of the source application, which can raise different events. However, the events that a source application can raise are common to the source application type. This means that a source application type has a specific set of events that can occur, which are specific to the type and version of the application.

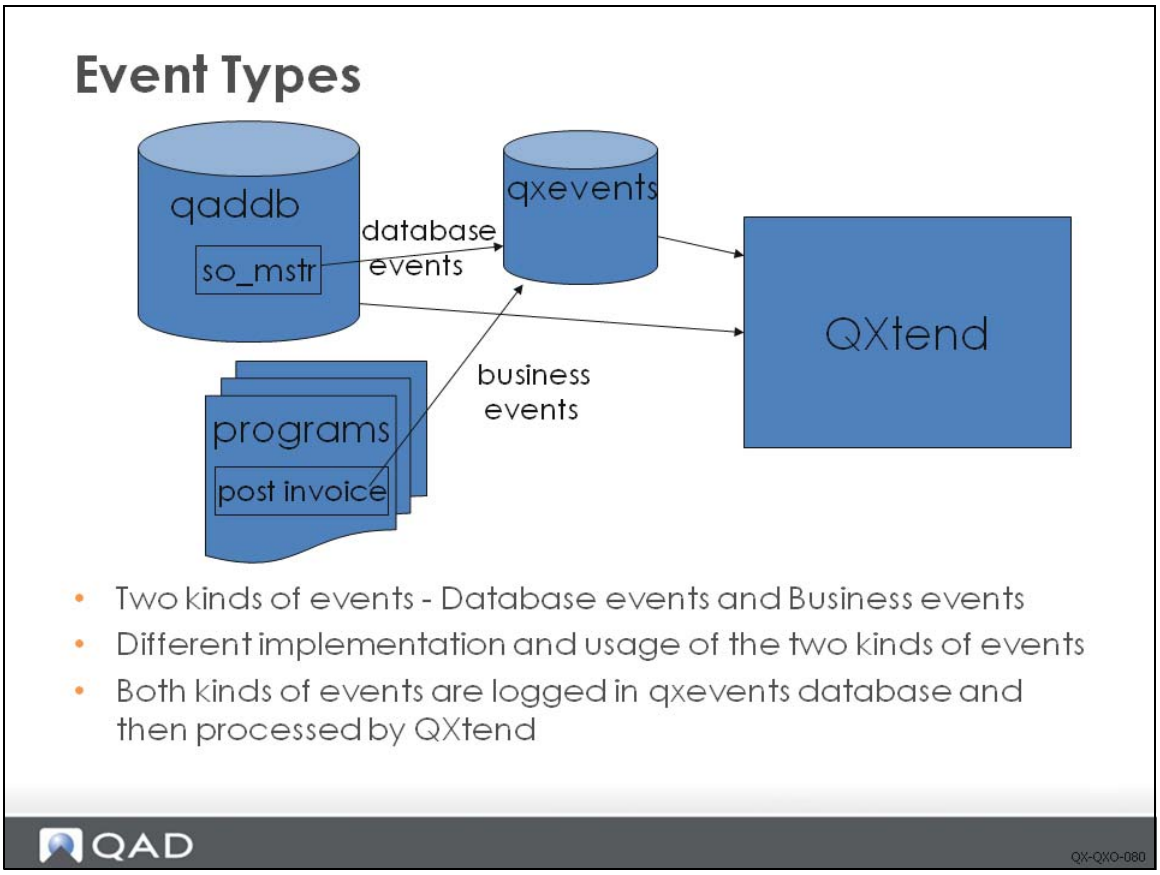
You manage events for a specific source application using the Event Types node under the Source Application node in the tree menu on the Configuration tab. The Event Types page lists all events that can occur in the source application. Only the events that have the Active check box selected are processed in the source application and by QXO.

The default list of event types installed with QXtend does not include all possible event types that could be required; QXO allows you to create new event types. QXO also allows you to modify or delete event types for a source application. When adding a new database table event, ensure that the replication write and replication delete triggers are set up to publish the changes that are made to the database table.

The Filter panel on top of the events lists can be used to search for specific event types.

By clicking the import button, events can be imported from the event file of the source application type which is resided in the events folder under the source application type folder.

By clicking the export button, events can be exported to the event file of the source application type (will overwrite the event file).



There are two kinds of events:

- Database Events (Database Triggers): Events raised by databases relate directly to changes in a specific database table. The event name must match the name of the table in the application database that has changed; most events are triggered from database schema triggers.
- Business Events (Named Events): Events raised from within the business logic, they have a meaningful names which reflect their business semantics and at the same time they are also linked to the affected database table name.

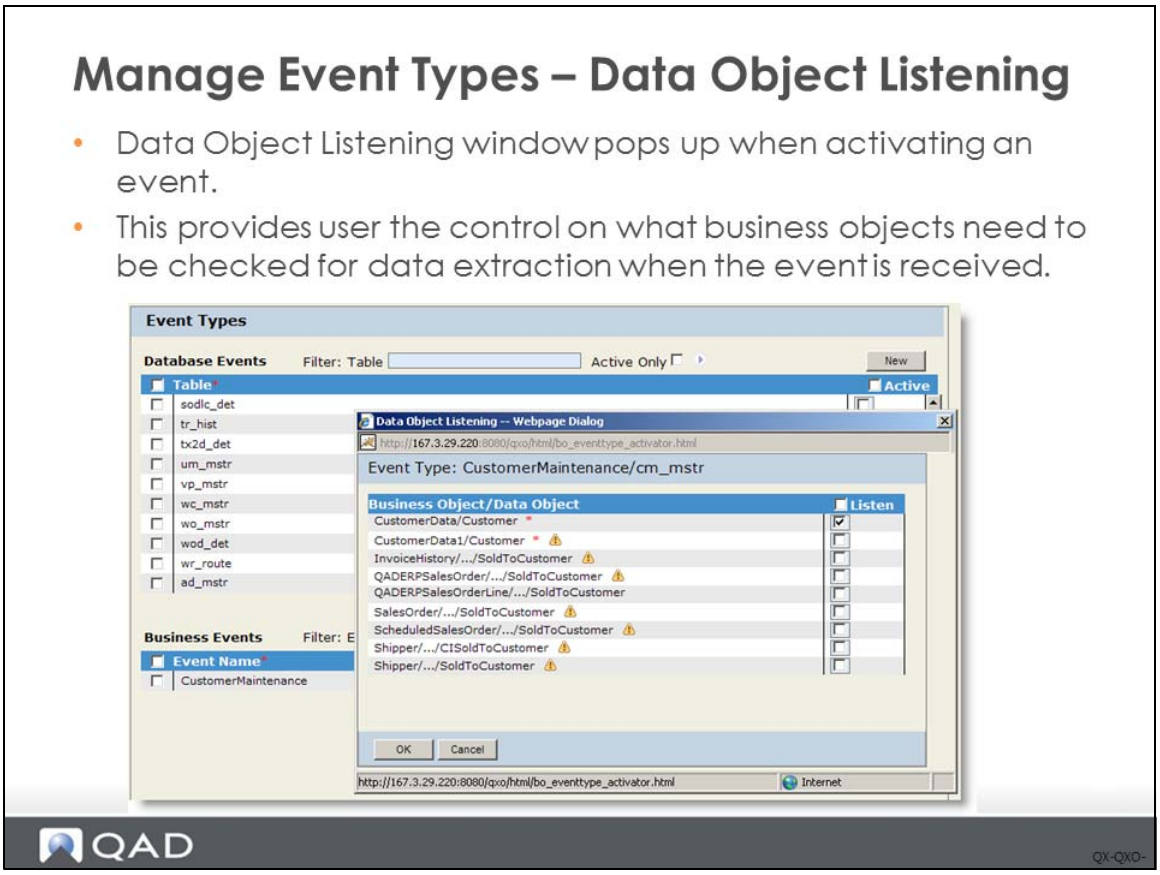
Business events enhance your control over when a business object is extracted, including the ability to trigger processing at crucial stages within a workflow.

For example, a sales order typically has a well-defined life cycle, from entering the sales order into the system, through processing the shipment, then printing and posting related invoices. If user only wants to extract sales order when related invoices have been posted, user can activate the InvoicePost business events and then QXtend can extract sales order data when this events is processed. This can hardly be implemented by using so_mstr database triggers as the database trigger will generate events for every data change to so_mstr table.

By default, QADSE and QADEE include some business events that are commonly used, like ItemMaintenance, SalesOrderMaintenance, etc. User can customize QAD Enterprise Application programs to add business events as needed. This is an advanced topic.

Comparing to QADSE, QADEE includes more business events. They are business events for QAD Financials business objects and typical event names for QAD Financials business objects are Create, Modify, Delete, and etc.

There is no default business events for eB, eB2, and eB21.

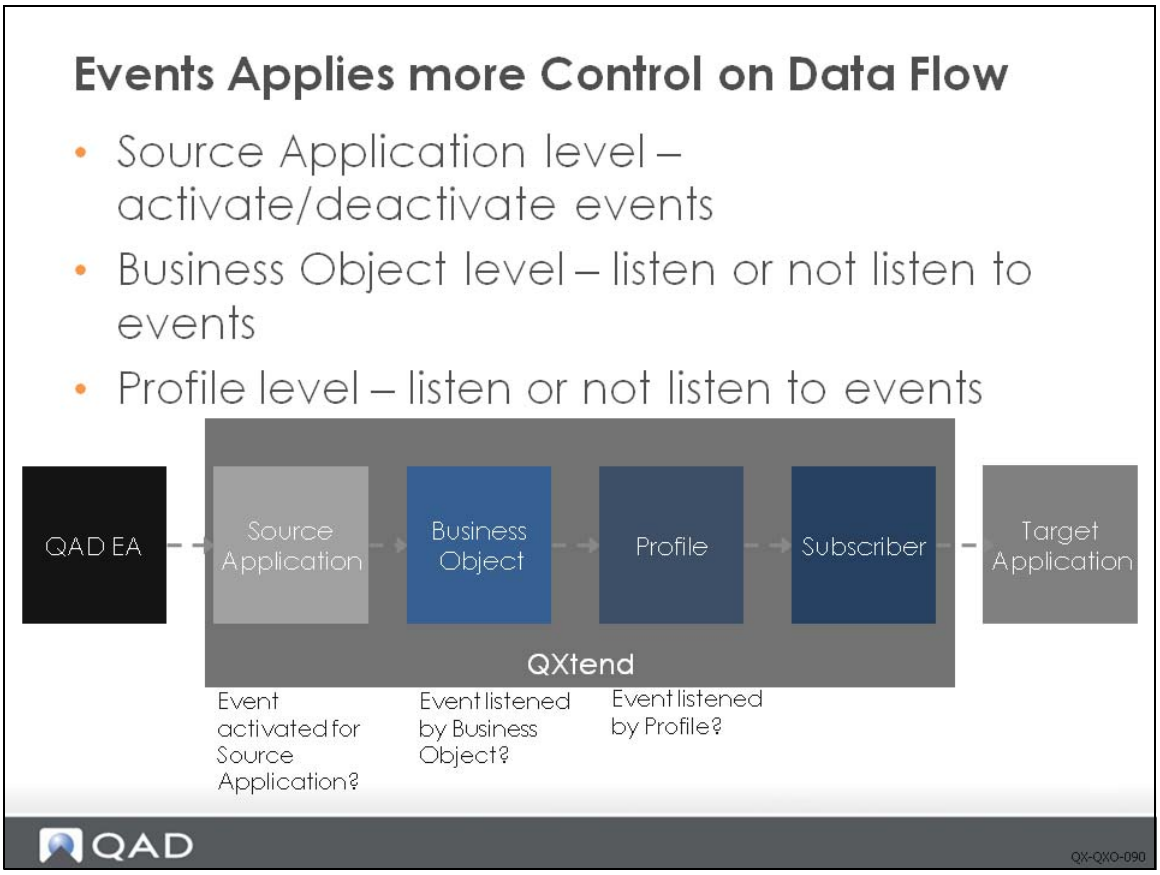


If you selected the Active check box, the Data Object Listening window pops up. The window displays all the business objects and data objects associated with the event type using the following formats:

- BusinessObject/DataObject * for data objects at the top level of business objects with the asterisk indicating that it is a top-level data object
- BusinessObject/.../DataObject for data objects at lower levels of business objects. To see the full path of a data object under the business object, move the mouse cursor over or double-click the abbreviated pathname.

Select the data objects you want to listen to the event type; then click OK.

You must activate the event types as well as configure the data objects to listen to the event types before any data extraction can take place. No data will be extracted for an inactive event type or an active event type with no data objects set to the listening mode. Also, do not activate any event types or turn on the listening mode for any data objects you do not plan to egister with any event service to be processed.



Events allow user to apply more control in data flow. For example, one external application is interested in any change to a sales order in QAD Enterprise Application, while another application is only interested in sales orders that have been posted. In this situation, we can use different event types (so_mstr and InvoicePost) to control the data delivery to different applications.

At the Source Application level, user can activate/deactivate events, as we just learned.

At the Business Object level, user can configure whether an activated event is listened by a business object.

At the Profile level, user can configure whether an activated event is listened by a profile.

Exercise: Source Applications

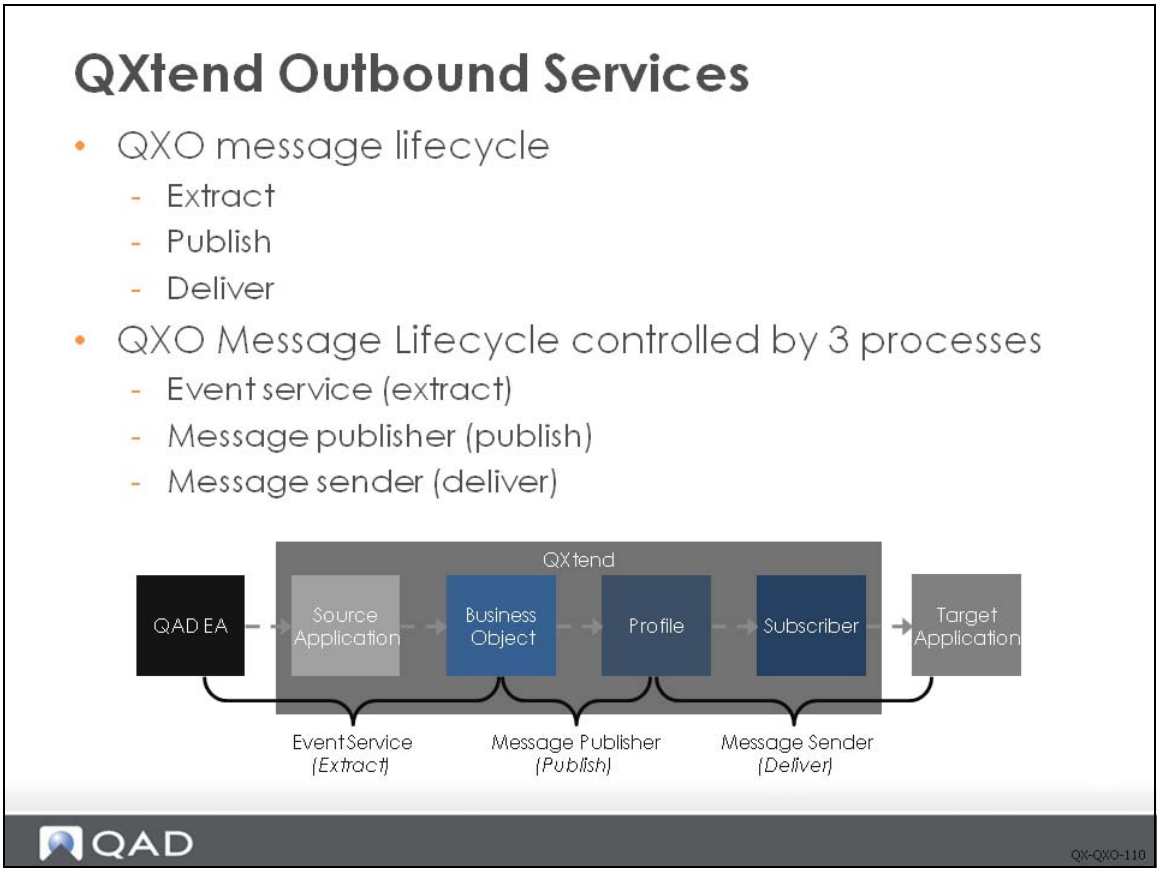


The following list shows a number of key concepts used in the source applications in QXO. In each statement below, fill in the correct term from the list.

qxevents	instances
application database(s)	source application
business events	events
name	mass rowid synchronization utility

1. _____ are raised by the _____ and identify when changes have been made.
2. The source application is assigned a unique _____ that identifies the instance.
3. Source application _____ are linked to a specific source application type.
4. When performing a dump and load, you can use the _____ to synchronize the rowids in QAD QXtend with the new rowids in the source application instance.
5. Source applications will always have a minimum of two databases defined: the _____ database and the _____ database.
6. The events raised from within the business logic are called _____.

Services



Taking an event that is raised in the source application and delivering the data about that event to interested applications has three phases. Each phase can be scaled to support high message throughput requirements. The three phases are:

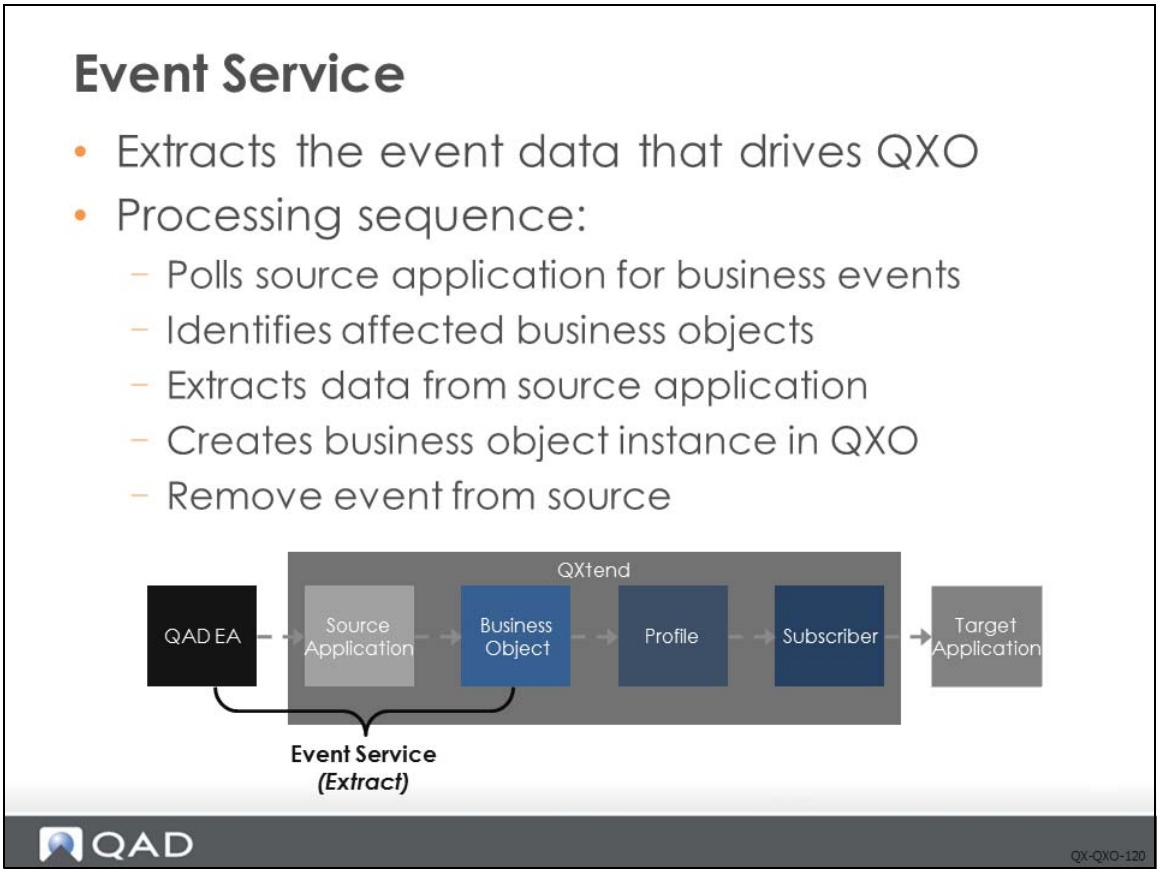
- Extract: Changes to data in a QAD Enterprise Application are extracted from the source application database using a format matching the structure defined in the QXO business object definition. Extracted data is stored in QXO as an instance of the business object.
- Publish: The stored business object instance data is mapped into the profiles defined against the business object. Any calculated fields and/or fixed values defined against the profile are calculated and/or assigned during this phase.
- Deliver: Profile messages generated in the previous phase are passed to each of the subscribers that have registered an interest in the profile message. The message is delivered to the target application using the details configured on the subscriber.

Outbound message processing services are responsible for moving a message through these phases. After a message has completed one phase, it is made available to the next service to take it through the next phase. The QXO services are:

- Event service: Responsible for the extract phase of the message life cycle. The event service retrieves source application events and identifies the business objects that need to be extracted. The event service uses the configuration details in the source application configuration to connect to the source and extract the necessary data. It then stores the data as a business object instance.

- Message publisher: Controls the publish phase of the life cycle. The message publisher identifies the next unpublished business object message and then identifies the profile's messages views that need to be created. The publisher creates each message and stores it for delivery.
- Message sender: Handles the delivery phase. The message sender locates the next message to be delivered and controls the process of sending the message to the QXO subscriber, which identifies the target application.

Event Service



The first phase of the QXO message life cycle is the *extract* phase, handled by the event service. In this phase application business events are monitored. The relevant application data related to the business event is extracted and stored in QXO.

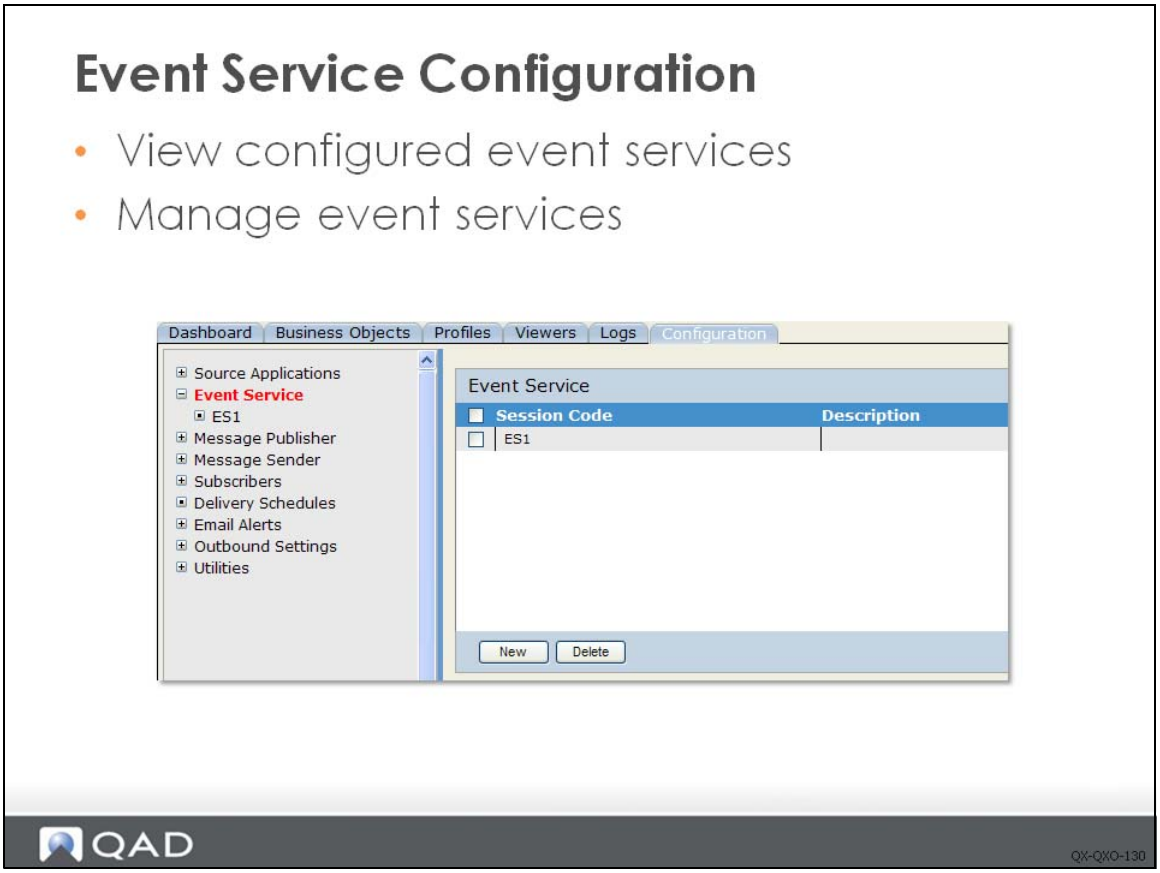
The event service extracts data from the data source and stores it in QXO using several steps:

1. Poll for event. Application events are logged by the source application into the `qxevents` database that is part of the QXO source application database set configuration. The events service uses the source application configuration information to connect to the `qxevents` database and searches for unprocessed events. If no events are present, it disconnects. When pending events are located, the event details are logged in the QXO database.
2. Identify affected business objects. All of the business objects that contain the table the application event was raised against are identified. Part of the identification process includes establishing whether or not extraction is required. To do this, the filters defined on the business object are evaluated. If the data does not meet the filter criteria, that business object is dropped from the affected business object list. The list of affected business objects drives the next step in the processing.
3. Extract data. Using the business object definition and the mapping from the business object to the tables in the source application, the business objects in the list built in step 2 are populated with data from the source application. Extraction uses native Progress ProDataset capabilities to load the data into the business object instance.

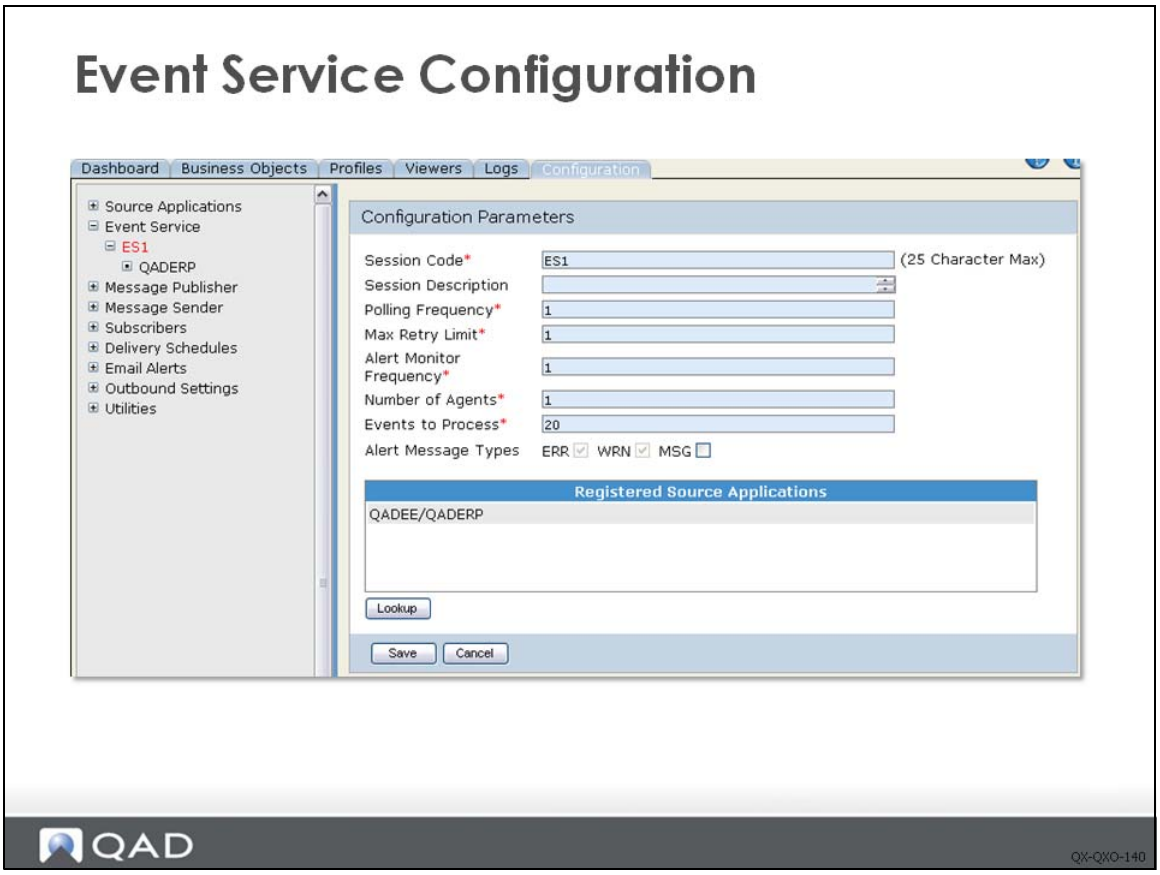
4 Create business object instance. The data extracted from the source application is compared to the previous message for the same business object instance. If none of the data contained in the business object has changed, no further action is taken—the event is ignored. If the application data has changed, the business object instance is stored in QXO as XML for processing by other QXO services.

5 Delete event. The application event in the data source is removed from the pending event queue by deleting the record.

Event Service Configuration



Event services are managed using the Configuration tab. You can view existing event services by selecting Event Service in the tree menu on the Configuration page. Click New or Delete to create or delete a selected event service. To modify an event service, select the event service name from the tree menu.



The Configuration Parameters screen of the Event Service allows you to define its behavior by completing the following fields:

Session Code. Enter a code (maximum 25 characters) for this service. Using a naming convention such as ES<*serviceName*> helps to identify the service as an event service.

Session Description. Enter a description.

Polling Frequency. Enter the number of seconds the service should wait between polls of the `qxevents` database. The `qxevents` database from the source application database set is polled to pick up any pending business events. The time delay specified here allows the event service to sleep before polling for pending events and prevents the event service from consuming valuable CPU resources.

Max Retry Limit. Enter the maximum number of times during a single session that the service will attempt to reprocess a failed application event (maximum 9999).

Alert Monitor Frequency. Specify the number of minutes between alert messages when the Alert Message Type is set to MSG and thresholds have been set.

Number of Agents. Specify the number of agents to start for the service when the service is started. Starting multiple agents increases the volume of messages that can be processed by the event service; in high volume environments, multiple event services can be started to meet the message throughput requirements.

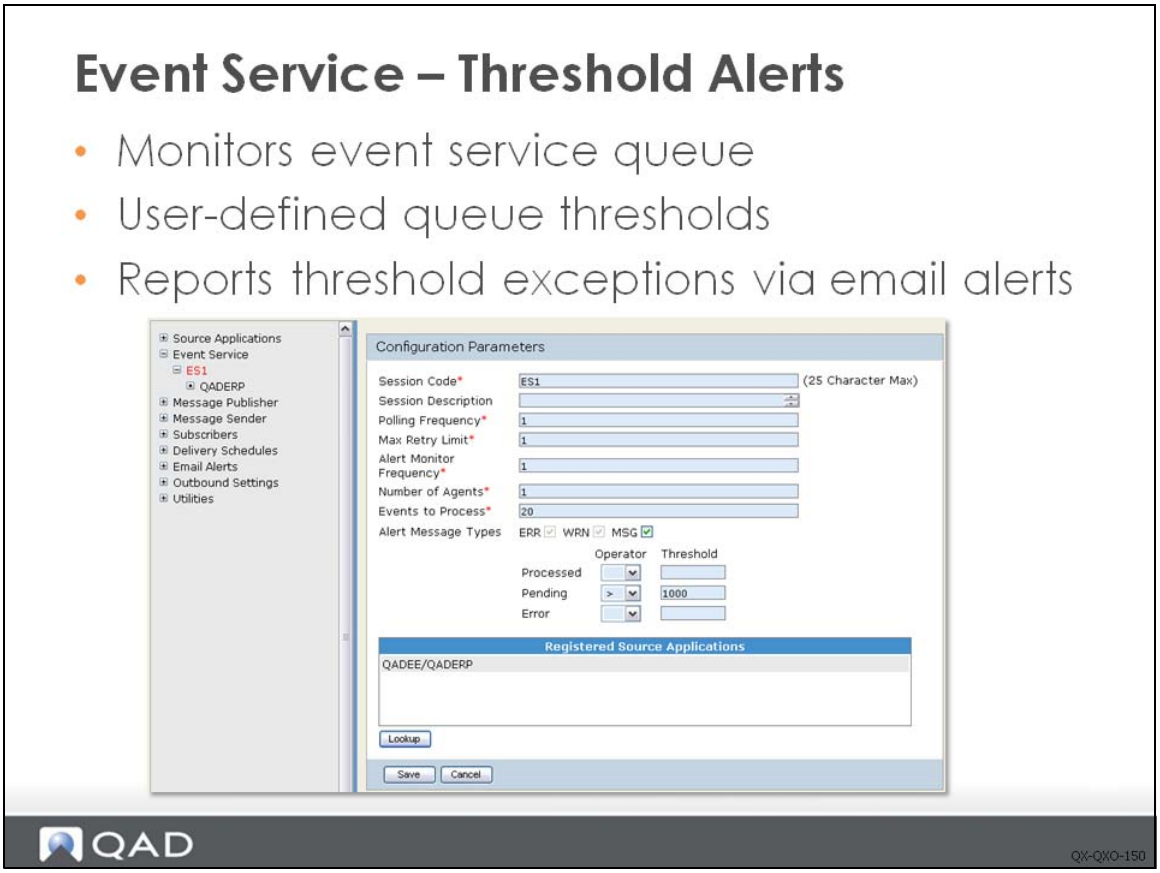
Events to Process. Enter the maximum number of events to be processed by a source application before switching to a different one. This is used only when an event service is defined with more than one source application, and is used to ensure throughput across all registered source applications. If message throughput across the source applications is not acceptable, you might want to configure an event service per source application.

Alert Message Types. Check the message types that you want to be raised as alert conditions for this profile. Choose one or all of the following:

- MSG (used for metric-related alert messages)
- ERR (errors)
- WRN (warnings)

The Alert Message Types setting works in conjunction with the email alerts functionality of QXO. The alert message types are only acted upon when email alerts are configured.

Threshold Alerts



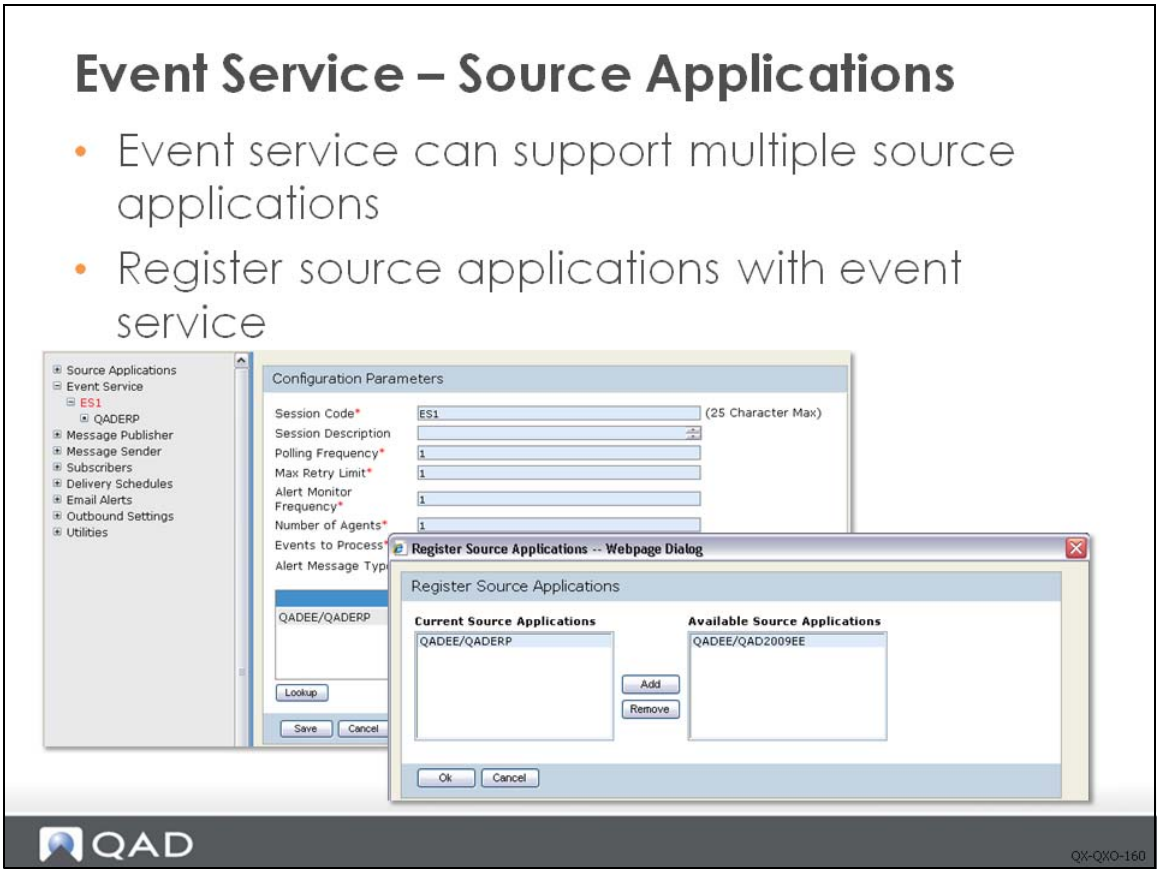
The event service processes events raised by the source application; this processing triggers other processes within QXtend that deliver data to subscribing applications. The data that passes through QXtend is often critical to the execution of the organization's business processes; therefore it is important that any bottlenecks in message processing are identified and alerts are raised. The threshold alerts enable the configuration of thresholds that can be used to identify potential problems early on, and help minimize the impact on the organization of any issues within QXtend.

To enable threshold alerts, select the MSG check box. You can then configure thresholds for the following:

- Processed: Monitors the number of messages processed by the event service.
- Pending: Monitors the messages waiting to be processed by the event service. This metric helps to identify how much work is backed up for the event service, and is important if there are critical messages backed up in the queue. You could start another instance of the event service to remove the backlog quickly.
- Error: Monitors the errors raised during event service processing. This can be used to identify potential problems that cause large volumes of errors to be raised.

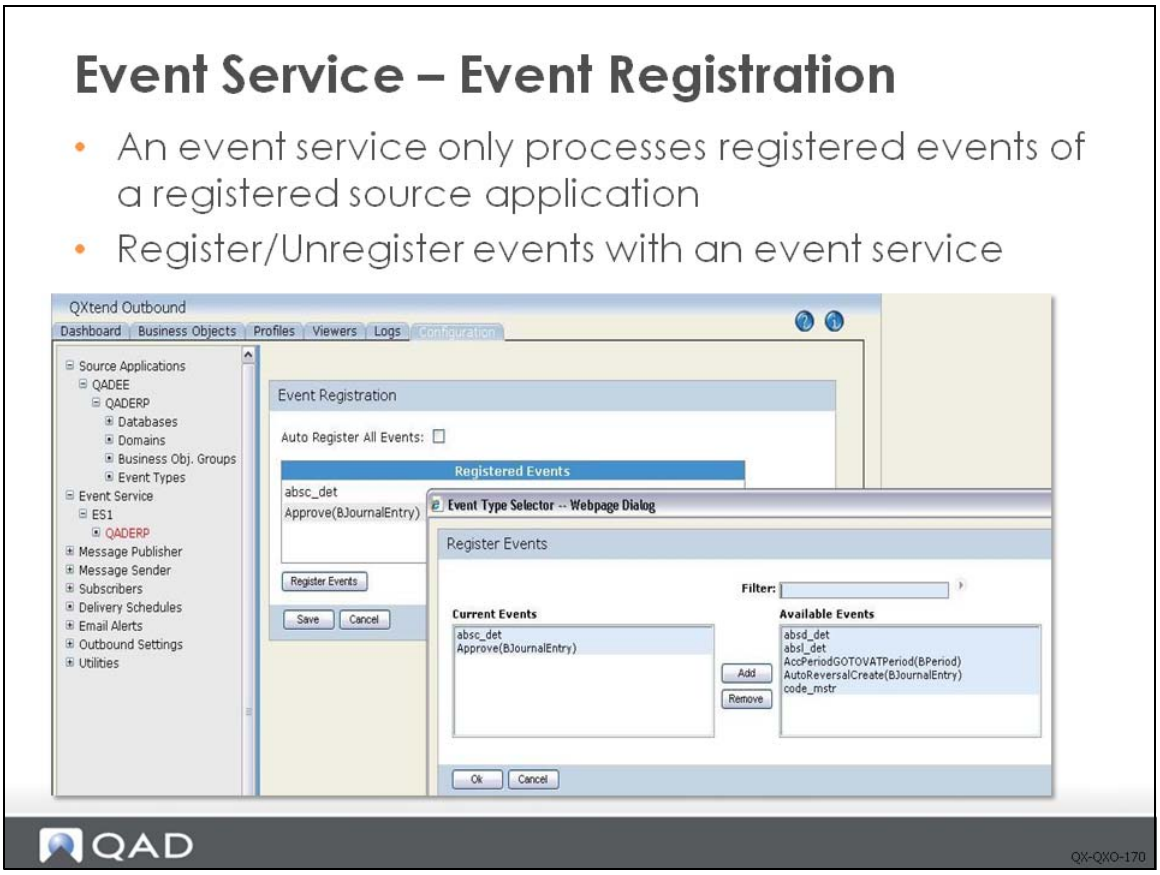
If a defined threshold is reached, an email alert is raised. Another alert will not be raised—although processing will continue—until the value specified in the Alert Monitor Frequency has elapsed. For example, if the Pending value is set to greater than 20 and the Alert Monitor Frequency is set to 10, if there are 100 messages pending, an alert would be raised. If 10 minutes later there are still more than 20 pending messages, another alert is raised.

Source Applications



A single event service can monitor events and extract data from one or more source applications. The source applications that an event works with are maintained in the Registered Source Application section of the event service Configuration Parameters page. Click Lookup to display the configured source applications, then select the source applications that the event service will process events for.

Note When adding multiple source applications to an event service, remember that the system will only process the number of events specified in the Events to Process parameter before moving on to service the next source application in the list. If a source application might have a high volume of critical messages, you should configure an event service that only services that source application.



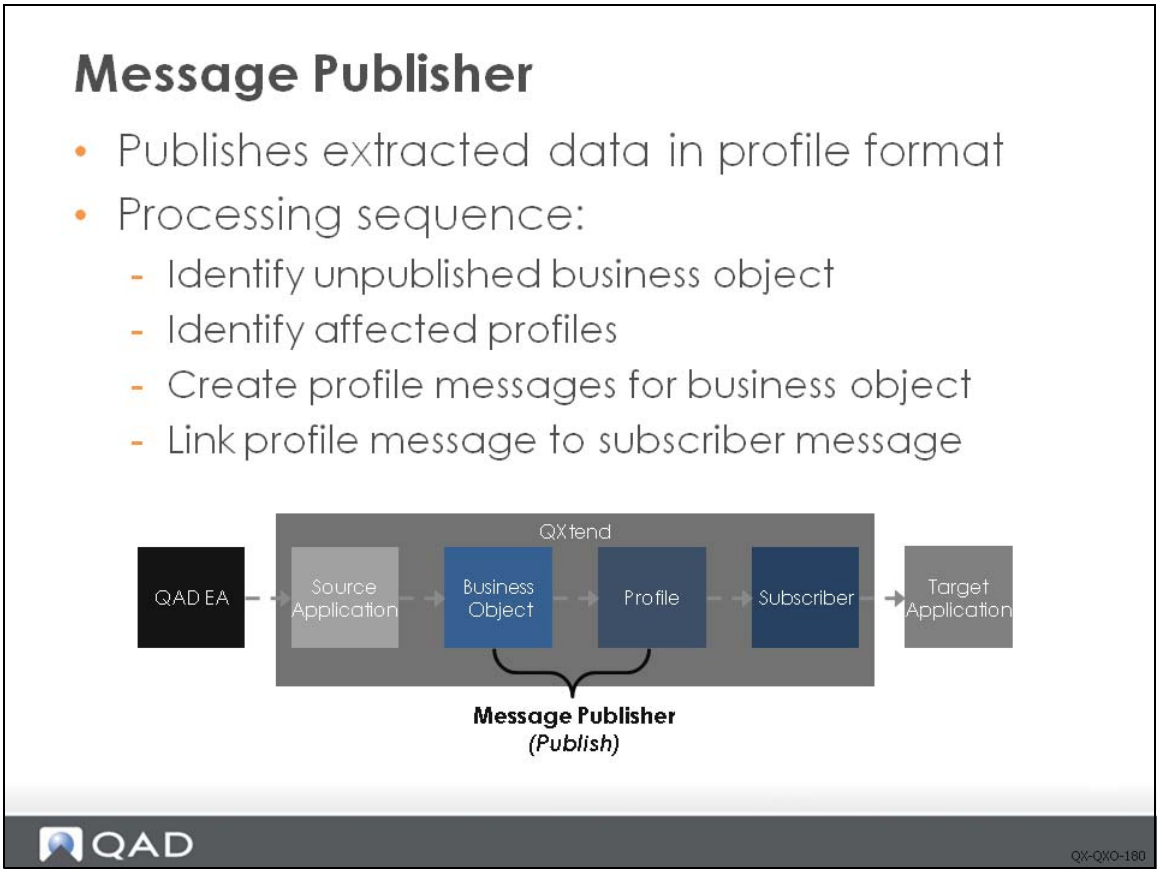
A single event service only process registered events of a registered source application.

By default, the Auto Register All Events checkbox is checked and all active events of the registered source application will be processed by the event service.

To register only some of the active events of a registered source application, clear the Auto Register All Events checkbox and click the Register Events button which will popup the Register Events dialogue within where current active events of the registered source application can be registered or unregistered with the event service.

Note When an event of a source application is deactivated it will be automatically unregistered from the related event services. When an event of a source application is activated it will be automatically registered to those event services where the Auto Register All Events checkbox is checked.

Message Publisher



The second phase of the QXO message life cycle is the publish phase, which is handled by the message publisher. In this phase unpublished business object instances are mapped into the relevant profile formats, and any calculated fields defined against the profile are evaluated.

The message publisher's purpose is simple: to publish data from the business object in the format required by the profiles that are defined against the business object, and then to store that data in QXO. To complete this process several steps are required:

1 Identify unpublished business objects

Business objects are extracted continuously; the message publisher looks for business objects that have not yet had the profile messages created for them. However, the message publisher only looks for business objects that it is responsible for publishing, which allows individual publishers to be created for high volume business objects, allowing a flexible, scalable infrastructure. The message publisher takes the first unpublished instance that is ready to be published and starts the publishing process.

2 Identify profiles

All of the profiles for the business object that are subscribers are identified. QXO does not create profile messages for profiles that do not have any active subscriptions. The publisher then takes this list of profiles and evaluates any filters defined against the profile. If the data in the business object does not satisfy the filter's criteria, the profile is dropped from the list. Finally a check is performed to see if any data on the profile has changed. Since the profile is a

subset of the business object, the business object might not contain any changes to the data in the profile. If no data changes are to be published, the profile is removed from the list, leaving the list of profiles that need to have messages created.

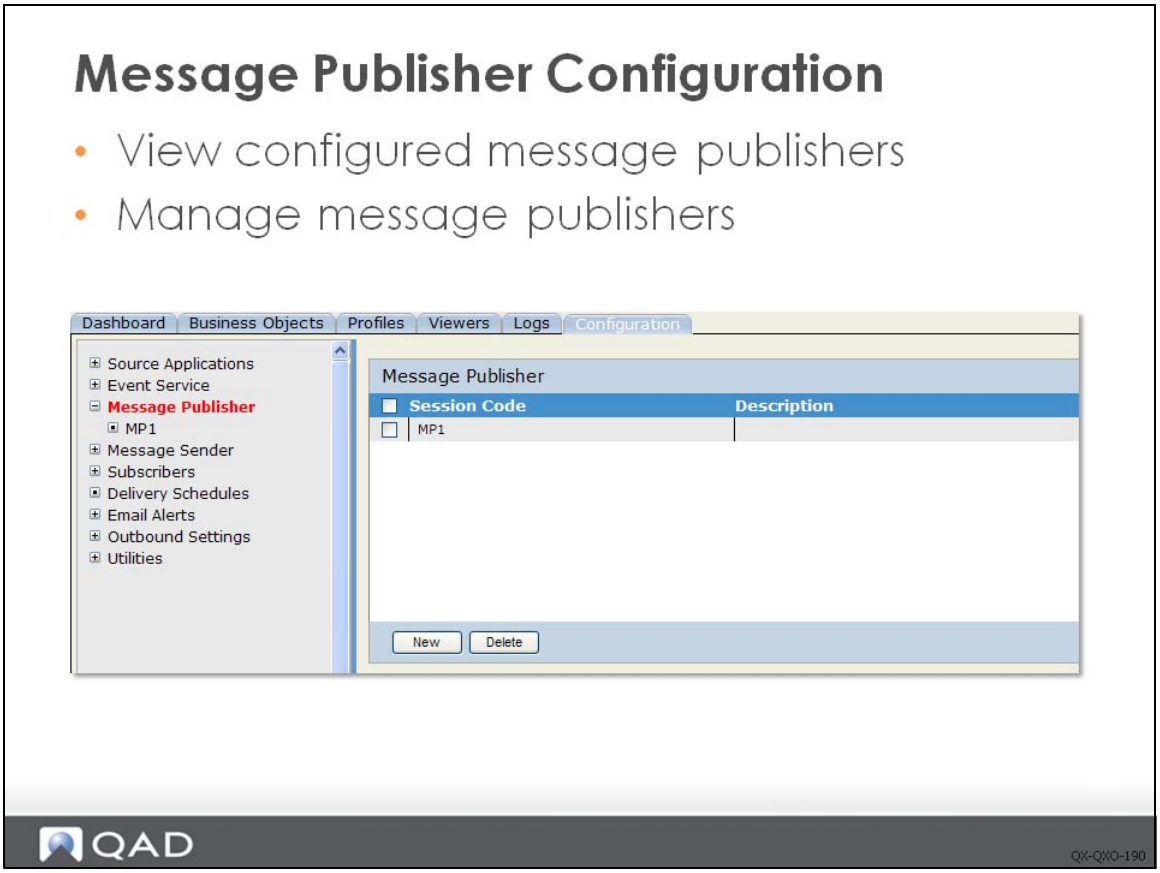
3 Create profile messages

Each profile identified in the list generated in Step 2 is created from the business object instance. As part of this process any calculated fields in the profile are evaluated. The profile is then stored in the QXO database as XML. If there is no profile message needs to be created from a business object instance (which is called raw message as well), the business object instance will be deleted.

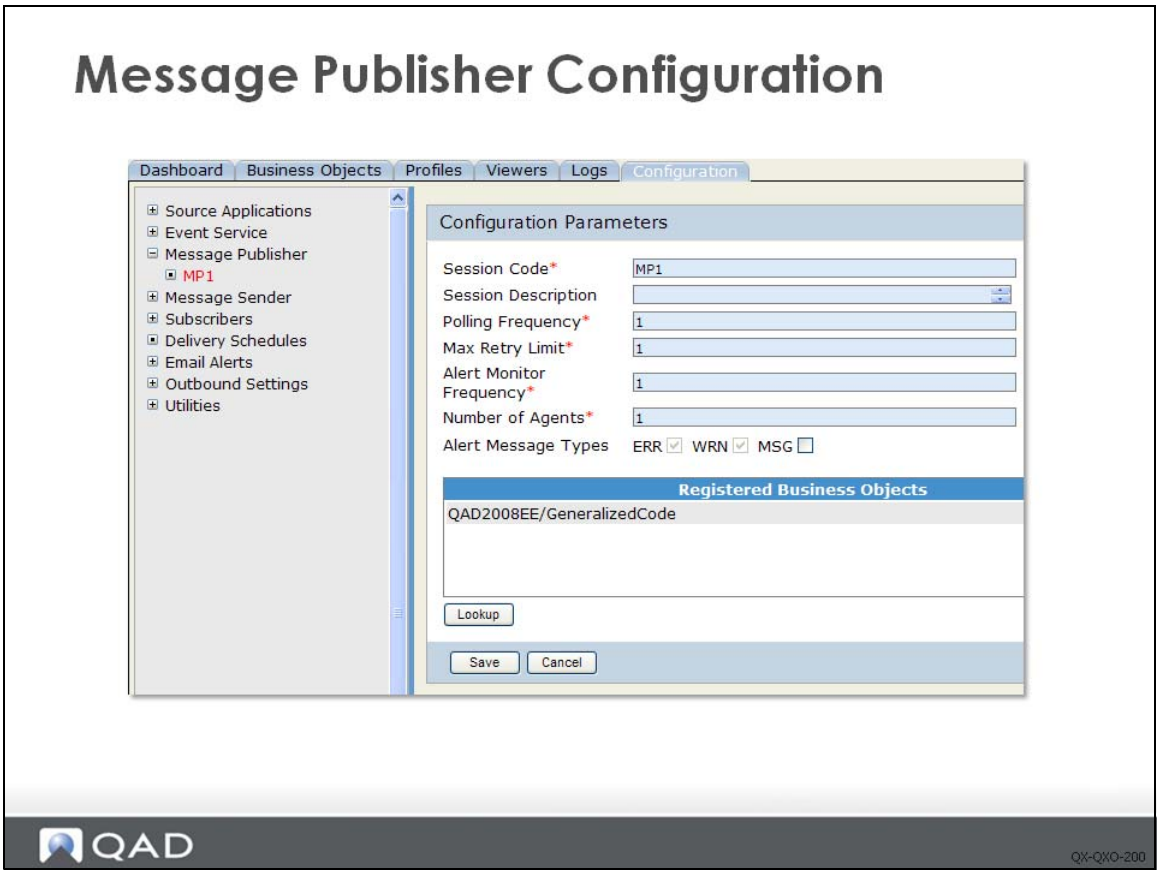
4 Link profile message to subscriber

In this step the system creates a link to the profile message for each of the subscribers that have registered to receive updates from the profiles created in Step 3. The link is stored in the Subscriber Messages queue, available in the Subscriber Messages viewer. This information is used by the sending process to determine which profile messages need to delivered, and to which subscribers. Once the messages are linked to the subscriber, they show as pending requests in the Subscriber Messages viewer page.

Configuring the Message Publisher



Message publishers are managed via the Configuration tab. To view existing message publishers, select the Message Publisher node in the tree menu. Use the New and Delete buttons to create and delete message publishers. To modify a message publisher, select the name of the message publisher from the tree menu.



Use the Configuration Parameters screen of the selected message publisher to define its behavior. Complete the following fields:

Session Code. Enter a code (maximum 10 characters) for this service. Using a naming convention such as MP<*serviceName*> helps to identify the service as an event service.

Session Description. Specify a description (maximum 15 characters).

Polling Frequency. Enter the number of seconds the service should wait between polling unpublished business object queue in the `qxodb`. The `qxo` database is polled to pick up pending business objects. The time delay specified here allows the message publisher to sleep before polling for pending business objects, preventing the message publisher from consuming valuable CPU resources.

Max Retry Limit. Enter the maximum number of times during a single session that the service will attempt to reprocess a failed application business object (maximum 9999).

Alert Monitor Frequency. Specify the number of minutes between alert messages when the Alert Message Type is set to MSG and thresholds have been set.

Number of Agents. Specify the number of agents to start for the service when the service is started. Starting multiple agent increases the volume of messages the Message Publisher can processed; in high volume environments, you can start multiple instances to meet message throughput requirements.

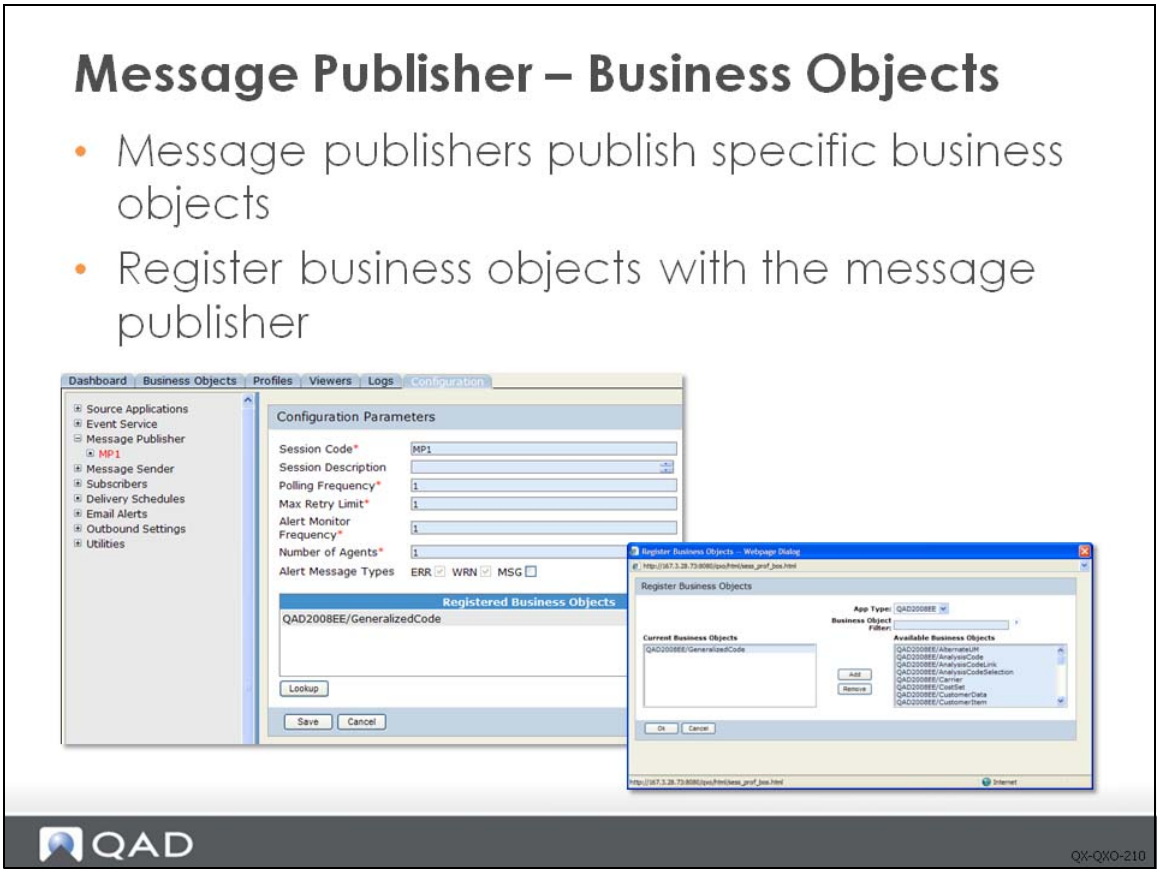
Alert Message Types. Check the message types that you want to be raised as alert conditions for this profile. Choose one or all of the following:

- MSG (used for metric-related alert messages)
- ERR (errors)
- WRN (warnings)

The Alert Message Types setting works in conjunction with the email alerts functionality of QXO. The alert message types are only acted upon when email alerts are configured.

Note The configuration of threshold alerts for the message publisher is the same as the event service. However, the threshold values apply to the number of profile messages being created.

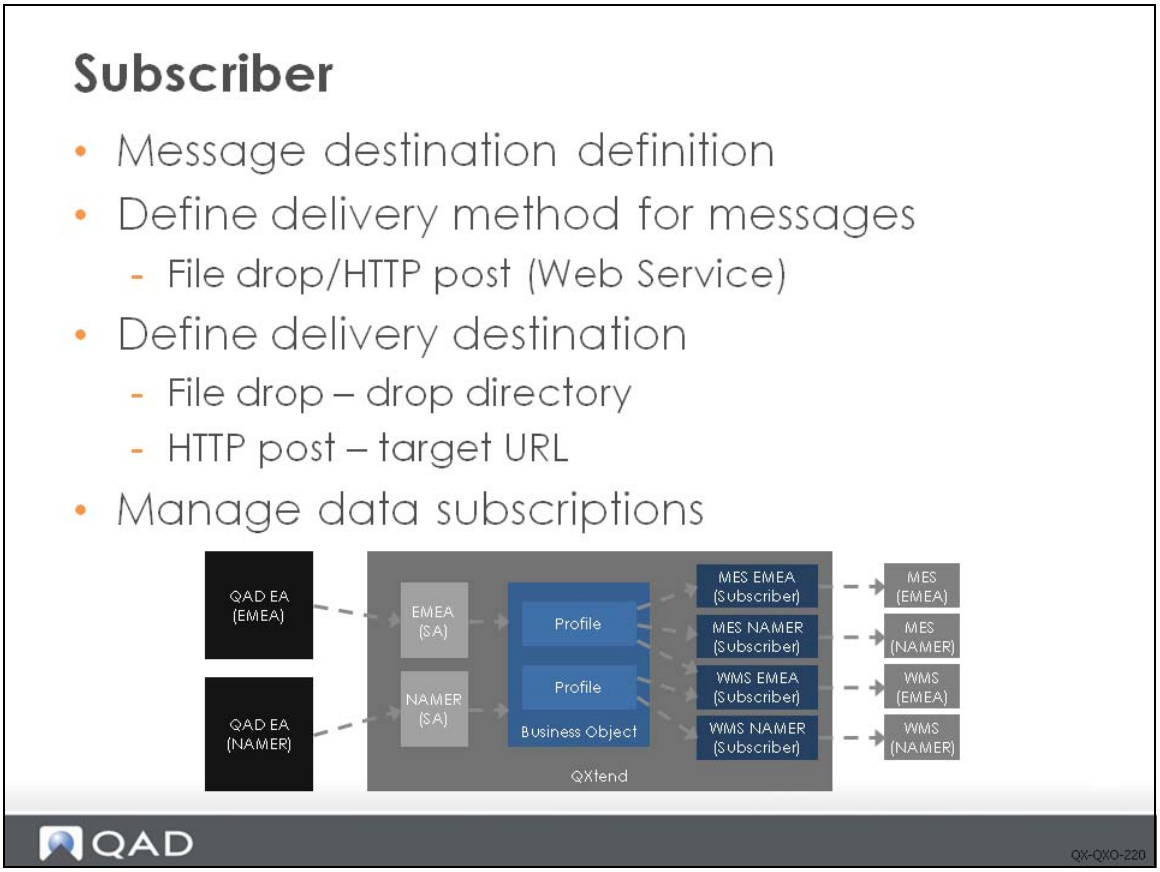
Business Objects



Message publishers work on a defined set of business objects that have been registered with it; this allows for flexible configuration during QXtend implementation. The message publisher publishes messages for business objects that are registered with it; this is useful if many messages must be processed quickly since having a dedicated message publisher ensures faster message processing. Messages that are low volume and non-critical can be grouped onto a single message publisher.

Click Lookup to display the selection box. Then select the application type, enter any filter for the business object search, and click the arrow next to the Business Object Filter field to execute a search; the business object list displays. Select the business objects you want to register, then click Add to register the business objects with the message publisher.

Subscribers

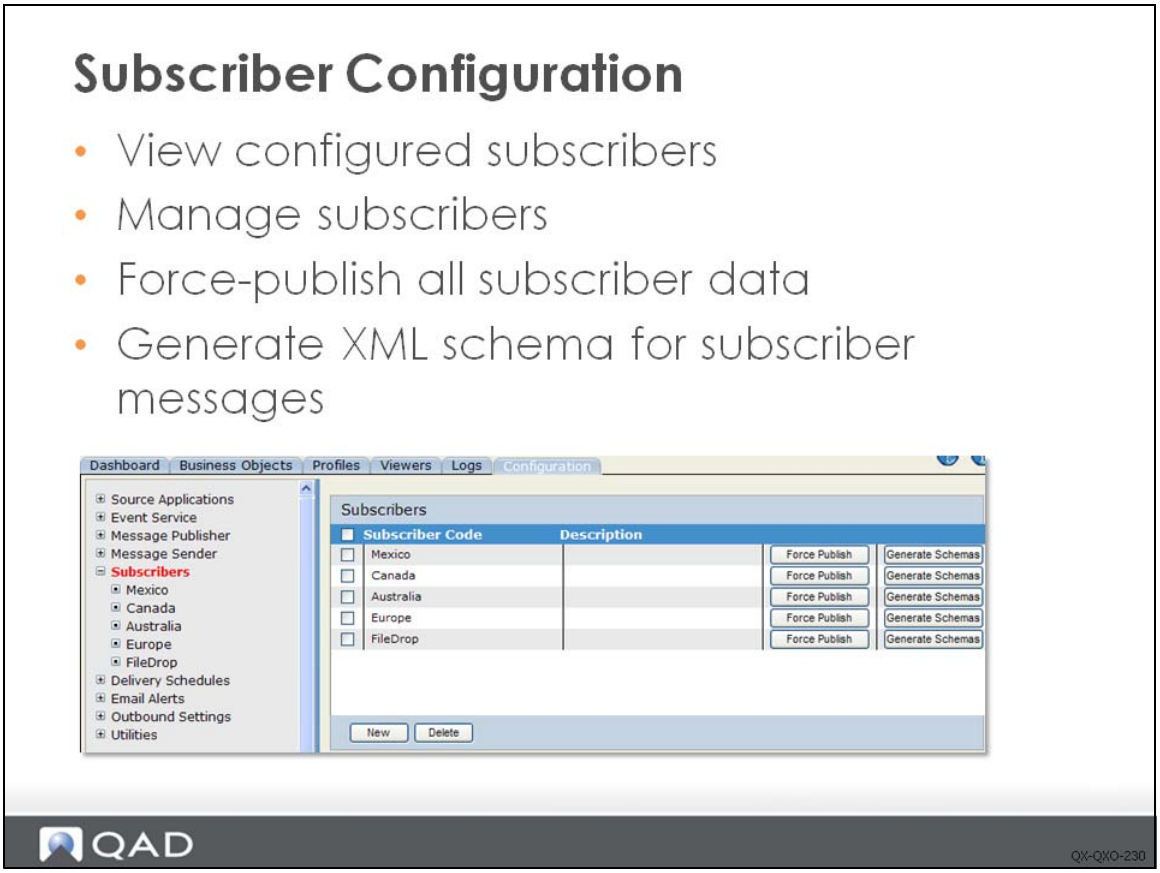


Before messages can be delivered, QXtend needs to know where to deliver the messages. *Subscribers* are used to define a message destination. To define a subscriber, the following configuration information needs to be provided:

- Delivery method: QXO can deliver messages to destination applications using either of two methods. The simplest method creates a file containing the published data in an operating system directory to which the server has write access; this method is called the *file drop* method. The other method (*Web Service* method) is essentially a simple `http(s)` post; the target of the post can be a Web Service, but the mechanics of the delivery are the same. The `http` post delivery method has two types: one type invokes the QXI Web Service; another type invokes any other Web Service or `http` post. These types are discussed later.
- Destination: For the selected delivery method, this defines the location where the message data should be delivered. The file drop delivery method requires that you provide the full path to the directory where the message should be created; this can be a path to a location on a network so long as the QXO server can access it. The Web Service delivery methods require you to provide the full URL for the service/`http` post; the URL must be accessible from the server and can use either `http` or `https` protocols.
- Data subscription: Destination applications (subscribers) are only interested in a certain set of data messages produced by QXO—they do not want to receive data that is of no interest. This is handled in QXO in the subscriber configuration by registering only the profiles that need to be delivered, ensuring that only required messages are sent.

QXO supports multiple source applications; however, a subscriber might only be interested in data from a subset of source applications. Subscriber configuration allows you to restrict the source applications that a subscriber receives data from.

Configuring Subscribers



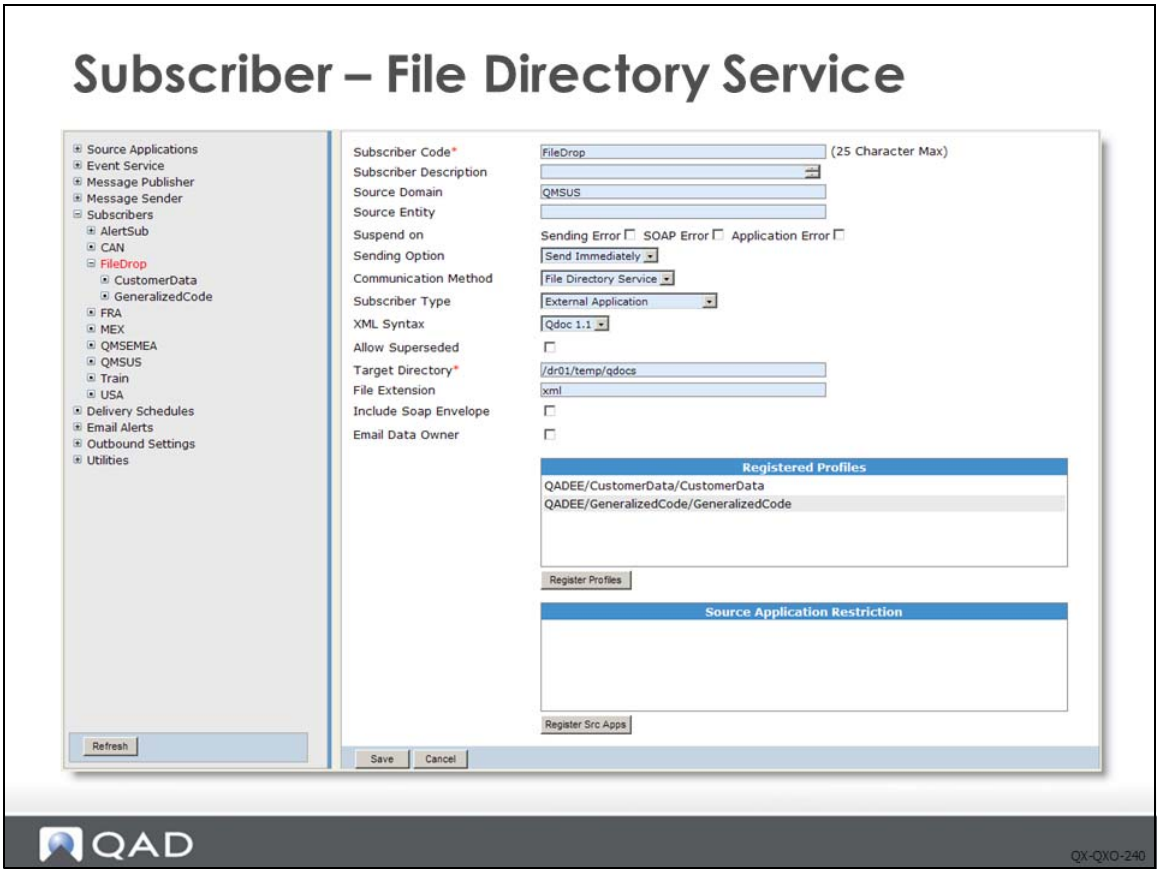
Subscribers are managed via the Configuration tab. You can view existing subscribers by selecting the Subscribers node in the tree menu. Use the Add and Delete buttons to create and delete subscribers. To modify a subscriber, select the name of the subscriber from the tree menu.

The Force Publish option allows you to force QXO to publish all data for some or all of the profiles registered with the subscriber. When the process is initiated, QXtend connects to the source application and creates events for all relevant data in the application; this forces QXO to process all data in the source application, synchronizing the data in QXO with the source application data.

Warning If the source application contains a large volume of relevant data, this could trigger many events that take a long time to compile; this is entirely dependent on the data in the source application.

The Generate Schemas option creates XML schemas that define each of the message types registered with the subscriber. XML schema define the content of the message, including the field data type that is published by QXO.

File Directory Service



The Configuration Parameters screen of the subscriber allows you to define the delivery method, destination, and data subscription along with other parameters that are required by QXtend or affect the processing of messages delivered to the subscriber. The configuration parameters are described below:

Subscriber Code. Enter a code (maximum 10 characters) for this subscriber. The name should clearly identify the subscriber and, if necessary, its location.

Session Description. Enter a description for the subscriber.

Source Domain. When data is published from a QAD application that supports the domain infrastructure, the domain from which the data was published is identified. This field is used if the subscriber is only interested in receiving data from a certain domain within the source application. Typically you might enter "`Master Data Synchronization`", where master data—such as customers, exchange rates—and so on are maintained in the "Master" domain and then replicated to other application domains. This allows the subscriber to receive updates only from the master domain.

Source Entity. Similar to the Source Domain parameter, this field is used with the QAD EE releases in which financial data is owned by an entity rather than a domain. This restricts messages that are sent to the subscriber to only those that originate from the specified entity.

Suspend on. Specify whether to suspend the subscriber on one or more of the following errors: sending error, SOAP error, and application error. Message Sender does not send messages to suspended subscribers and suspended subscribers can be resumed later through the UI or a batch process script.

Sending Option. The two possible values are Send Immediately or Use Delivery Schedule.

- Send Immediately requests that messages are delivered to the subscriber as soon as possible, there will be a delay depending on the number of pending message but messages will be sent at the earliest possible time.
- Use Delivery Schedule allows you to define a schedule that determines when messages are delivered—delivery schedules are user-defined. When using a delivery schedule, messages are marked with an earliest sending time to determine when a message is eligible for delivery. The message is not sent until that date and time has passed.

Delivery Schedule. This option only displays when Sending Option is set to Use Delivery Schedule. You can select a defined delivery schedule from the drop-down list. Messages for this subscriber will be delivered using the selected schedule.

Communication Method. Defines the delivery method to use when sending the message. The three methods are File Directory Service, Web Service, and QXtend Web Service. This example above is specific to the File Directory Service.

Subscriber Type. Specify the type for the subscriber. The options in this list represent the valid applications that have been registered with QXI. The combinations of communication method and subscriber type decides permission of using free messaging.

XML Syntax. Controls the structure of the XML message generated by QXO. Three formats are supported: QDoc 1.0, QDoc 1.1, and Other.

- QDoc 1.0 is currently a deprecated format, supplied only for backward compatibility.
- New implementations that generate QDocs should use QDoc 1.1, which controls the structure of the application data and the structure of the SOAP message.
- Other allows the SOAP envelope that is created to be flexible and does not create QDoc-specific elements that are created by the other XML Syntax options. Using this option, any SOAP message will not follow QDoc standards.

Note Choosing a QDoc syntax causes other fields to display. These are discussed later.

Allow Superseded. Allows subscriber messages to be superseded by subsequent instances of the subscriber message if the new message is created before the previous message has been delivered to the subscriber. The superseding message will contain all of the data changes present in the superseded message. This option can reduce the number of messages delivered to the subscriber, especially when used with a delivery schedule. Messages that are superseded are marked with a SUP status in the Subscriber Message viewer.

Target Directory. Only requested when the communication method is File Directory Service, enter the path to the directory where the outbound messages need to be created.

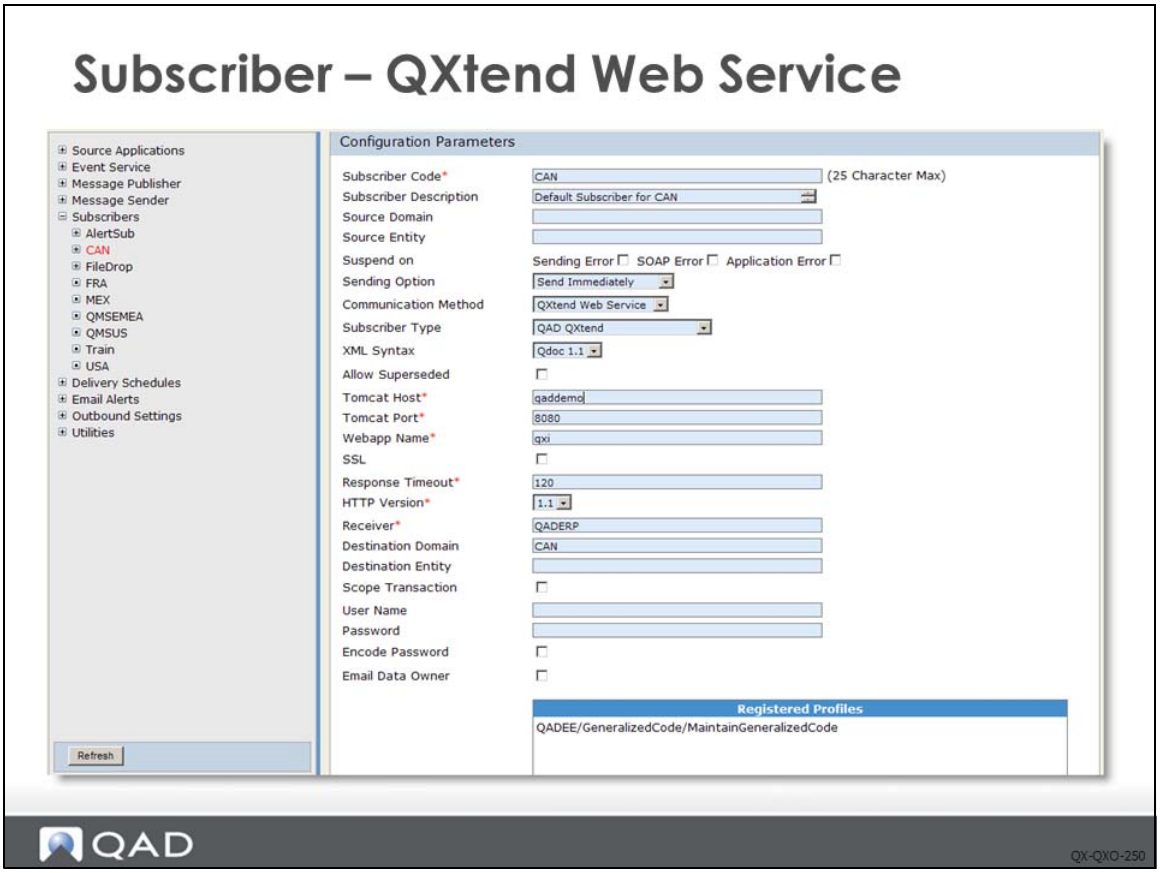
File Extension. Specify the extension of the output file. By default, it is “xml”, but this can be changed for particular purpose. For example, if “req” is specified as the extension, then the output files can be directly dropped into a queue directory to be processed by Inbound Queue Manager.

Include Soap Envelope. When messages are delivered to a Web Service, the application data must be part of a SOAP envelope. Selecting this option creates a QDoc envelope if the XML syntax is QDoc, and a user-defined envelope if the type is Other.

Note Choosing this option causes other fields to display. These are discussed later.

Email Data Owner. Select this check box to indicate e-mail should be sent to the owner of the data when a message raises an exception.

QXtend Web Service



You use the same Configuration Parameters screen to create a subscriber to deliver messages to QXtend Inbound. However, you set the Communication Method to QXtend Web Service. Several extra fields display that you must complete:

Tomcat Host. The name or IP address of the machine where the QXI instance that you want to process the request is running.

Tomcat Port. The port number of the Tomcat instance.

Webapp Name. The name of the QXI web application. This is the name assigned to the QXI web application during the installation of QXtend.

SSL. Select this option when the Tomcat instance that is hosting QXI is only accessible via SSL communication.

Response Timeout. The amount of time in seconds that QXO waits for a response from the HTTP post. If a response is not received in the specified time, QXO raises an exception.

HTTP Version. The version of the HTTP protocol to use when processing the HTTP post. Use HTTP post 1.1 unless you experience problems.

Receiver. The name of the receiver in QXI you want to process the request. The value entered in this field must match exactly, including the case the name of the receiver set up in QXI.

Destination Domain. The name of the domain that you want the message to be loaded into. If the destination domain is not specified on the subscriber, the default domain on the receiver connection pool is used.

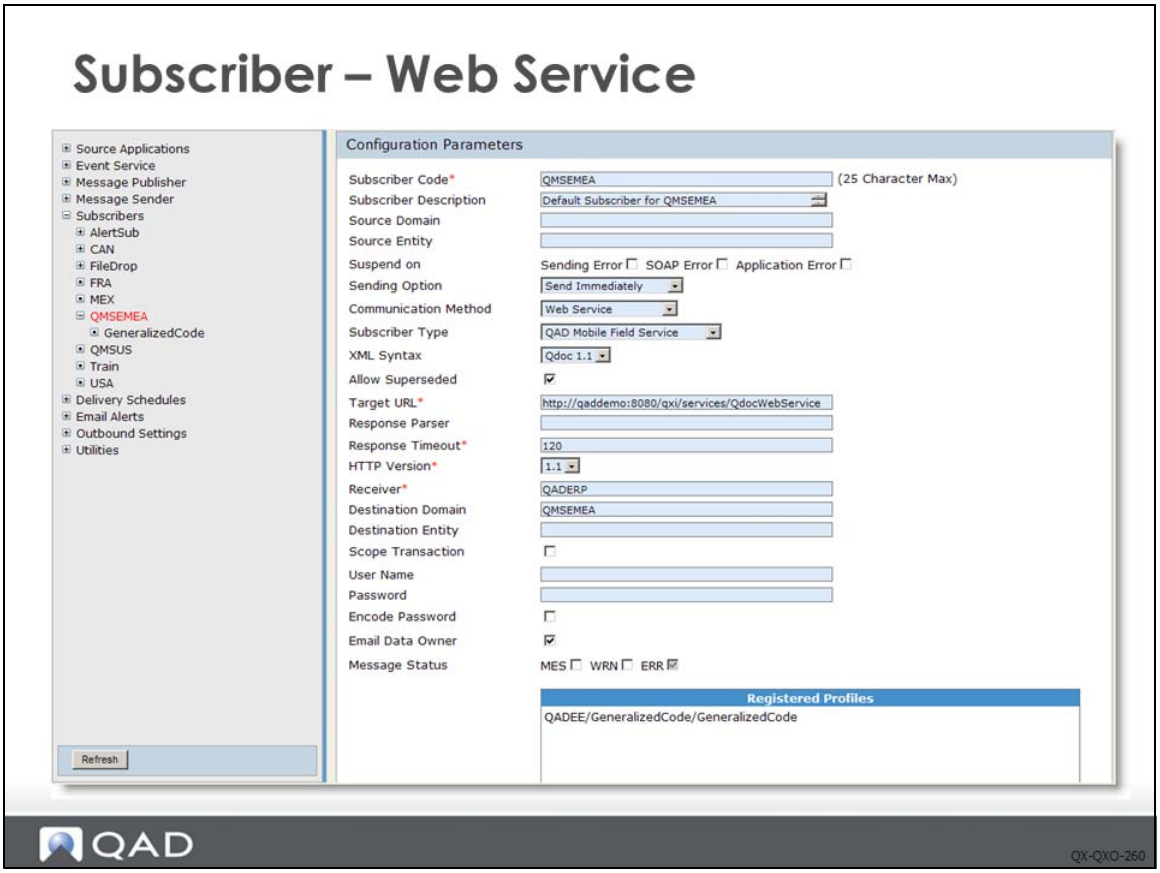
Scope Transaction. Select this option to indicate that the entire message should be treated as a single transaction, and that any exception should cause the entire transaction to be rolled back. If not selected, the normal transaction scoping applied by the application is enforced.

User Name. Name of the user to be used to process the transaction. This field is used by QXI only when the Use Requestor feature is selected.

Password. The password used to log in with the user name specified above. This field is only used by QXI when the Use Requestor feature is enabled.

Encode Password. Specify whether you want to encode the password for the subscriber.

Web Service

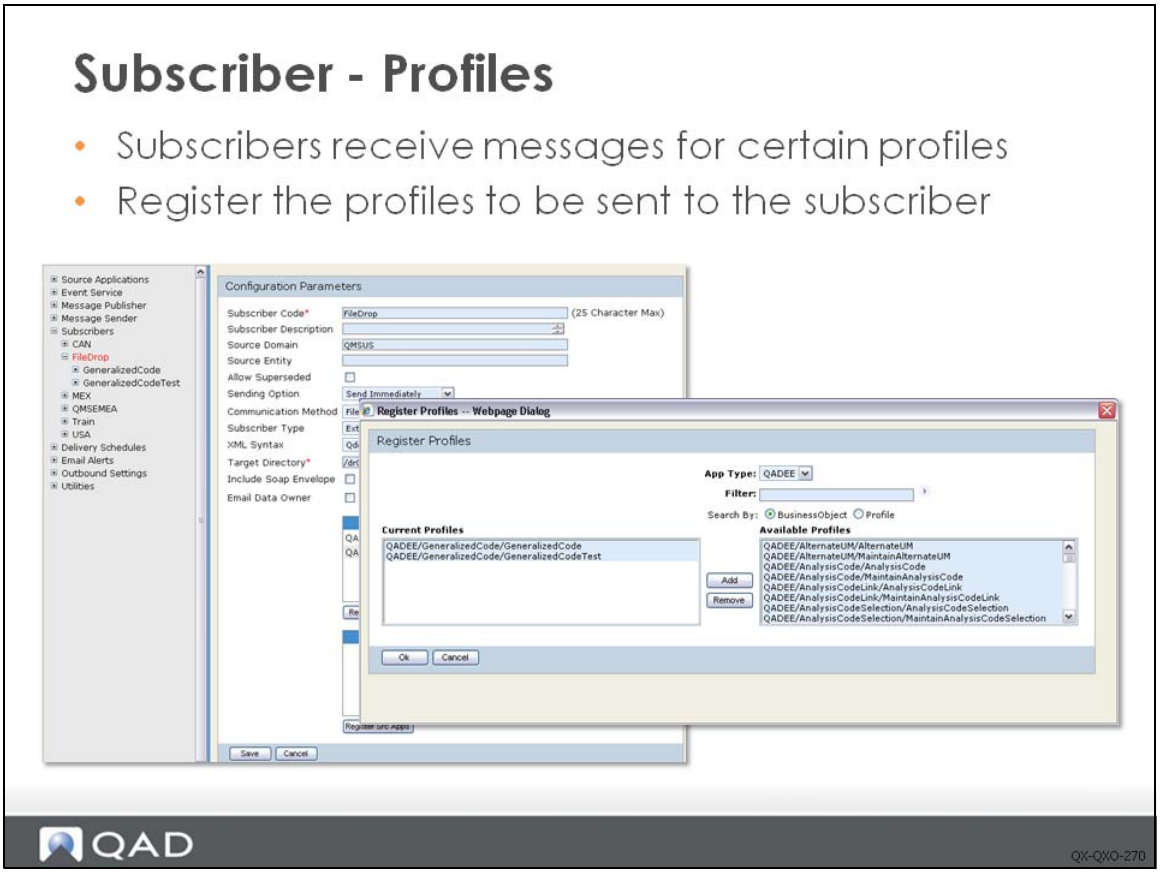


When creating a subscriber to deliver messages to a Web Service, you use the same Configuration Parameters screen. However, you set the Communication Method to Web Service. The configuration parameters are similar to Qxtend Web Service with the exception that you need to provide Target URL instead of Tomcat Host/Port/WebApp Name, etc.:

Target URL. The complete URL used to deliver the message to the target application.

Response Parser. Specify the response parser the subscriber uses to parse QDoc information. Response parser is a Progress class that you can define to parse information for your subscriber application.

Profiles

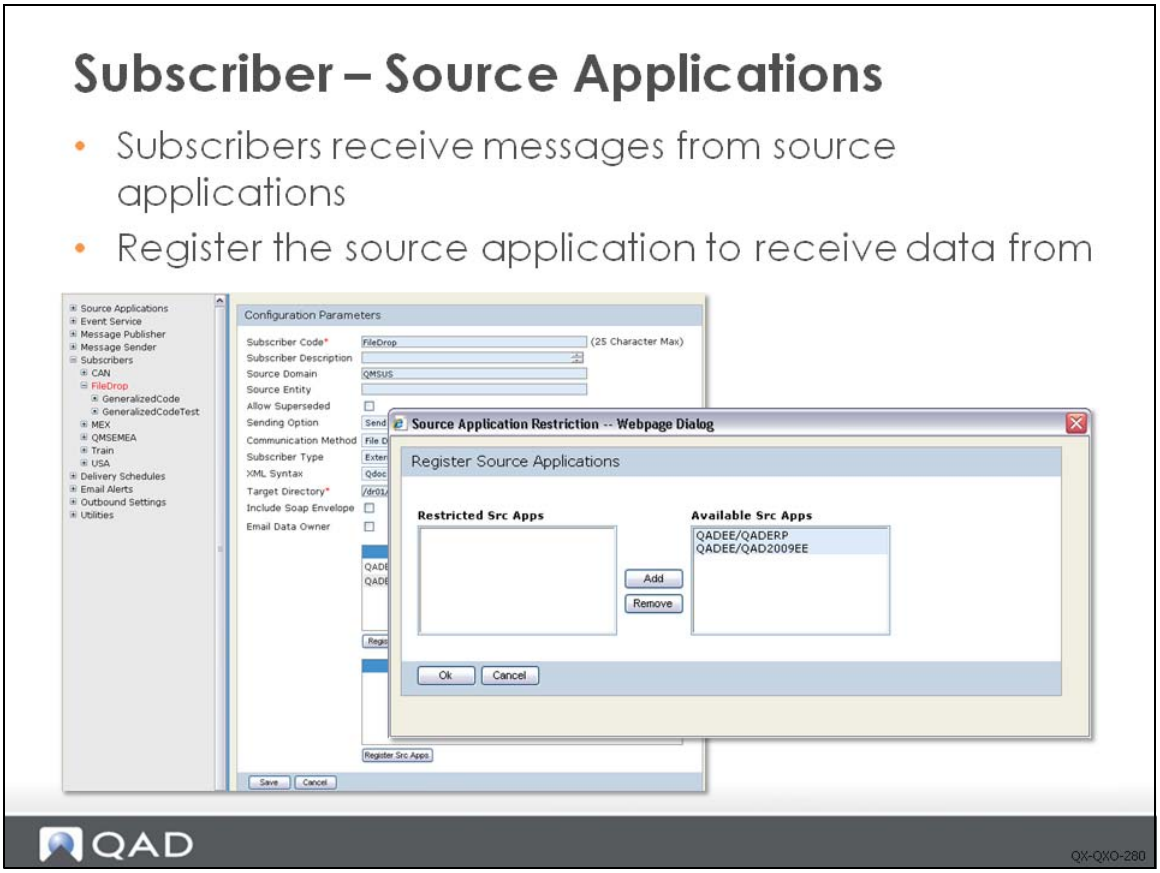


Subscribers are interested in only certain messages produced by QXO; the messages produced by QXtend are represented by the profiles available in QXO. Configuring the subscriber requires that you register the profiles to be delivered to the subscriber. This allows you to define which messages are delivered to the subscriber.

To register profiles:

1. Click Register Profiles to display the selection box.
2. Select the App Type, enter any filter for the Profile search, and then click the arrow next to the Filter field to execute a search and display the business object list.
3. Select the profiles you want and then click Add to register them with the subscriber.

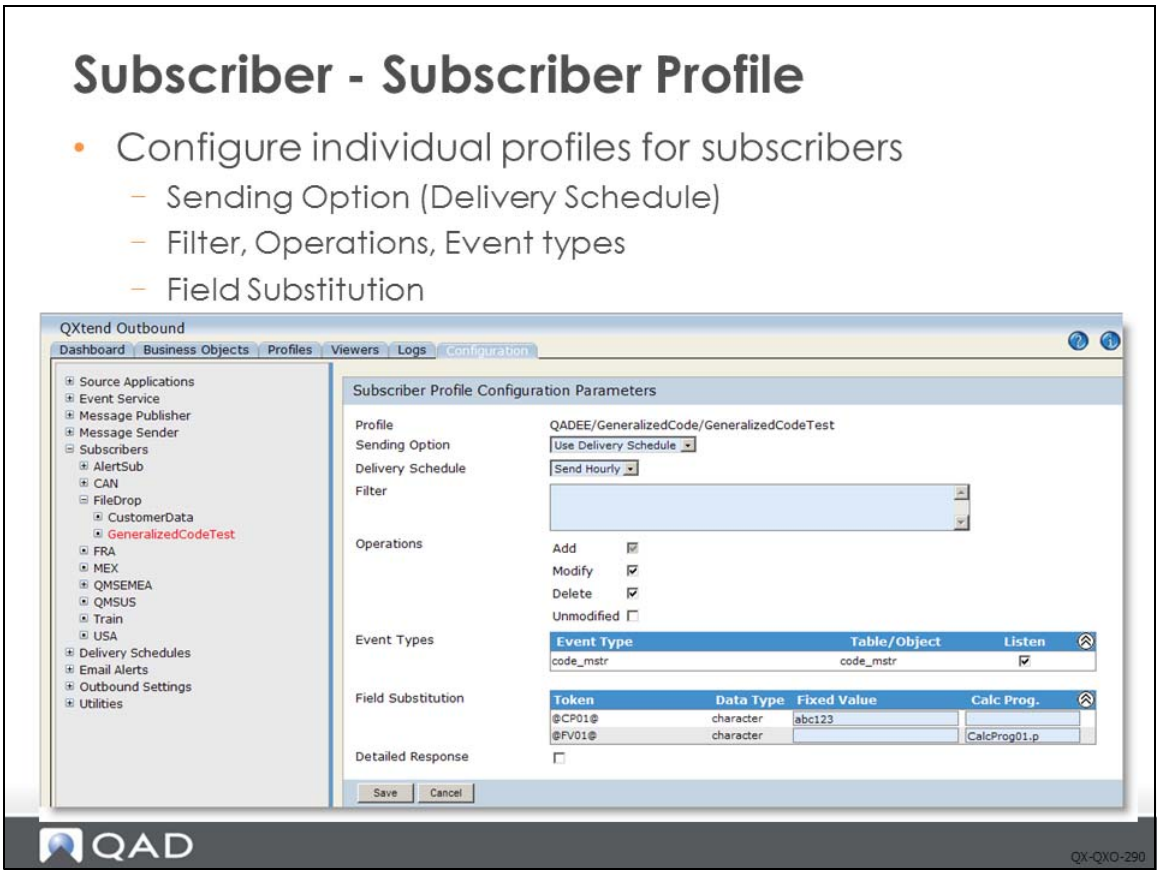
Source Applications



Subscribers receive data from source applications. QXO can support many source applications, but typically a subscriber wants to receive data from some, not all, source applications in QXO. To address this, you can restrict the source applications the subscriber receives data from. Each subscriber has a Source Application Restriction list; only messages from those source applications are delivered to the subscriber.

To display the selection box, click Register Src Apps, select from the Available Src Apps list, and then click Add to register them with the subscriber.

If no Src. App is registered with Source Application Restriction, then messages from all source applications can be delivered to the subscriber.



You now can configure individual profiles for subscribers to enhance your control over when messages are sent.

Sending Option. Choose a sending option to use when sending messages:

- Use Subscriber Default: Send the document to the subscriber using the delivery schedule that is defined for the subscriber.
- Send Immediately: Send the document to the subscriber immediately. If there are no delivery schedules defined, this is the default.
- Use Delivery Schedule: This option is available only if delivery schedules have been defined. Selecting this option displays a list of the defined schedules.

Filter. The subscriber will only deliver the profile message if the filter defined on the subscriber is true for the data in the top-level table in the profile.

Operations. When the message publisher determines the operation of the top-level buffer, the system will not create a subscriber message if the operation is invalid for the subscriber. If the Detect Operations check box is not selected on the profile, the message publisher determines the operation based on the existence of the previous and current message. Before creating the subscriber message record, the filter query must be satisfied.

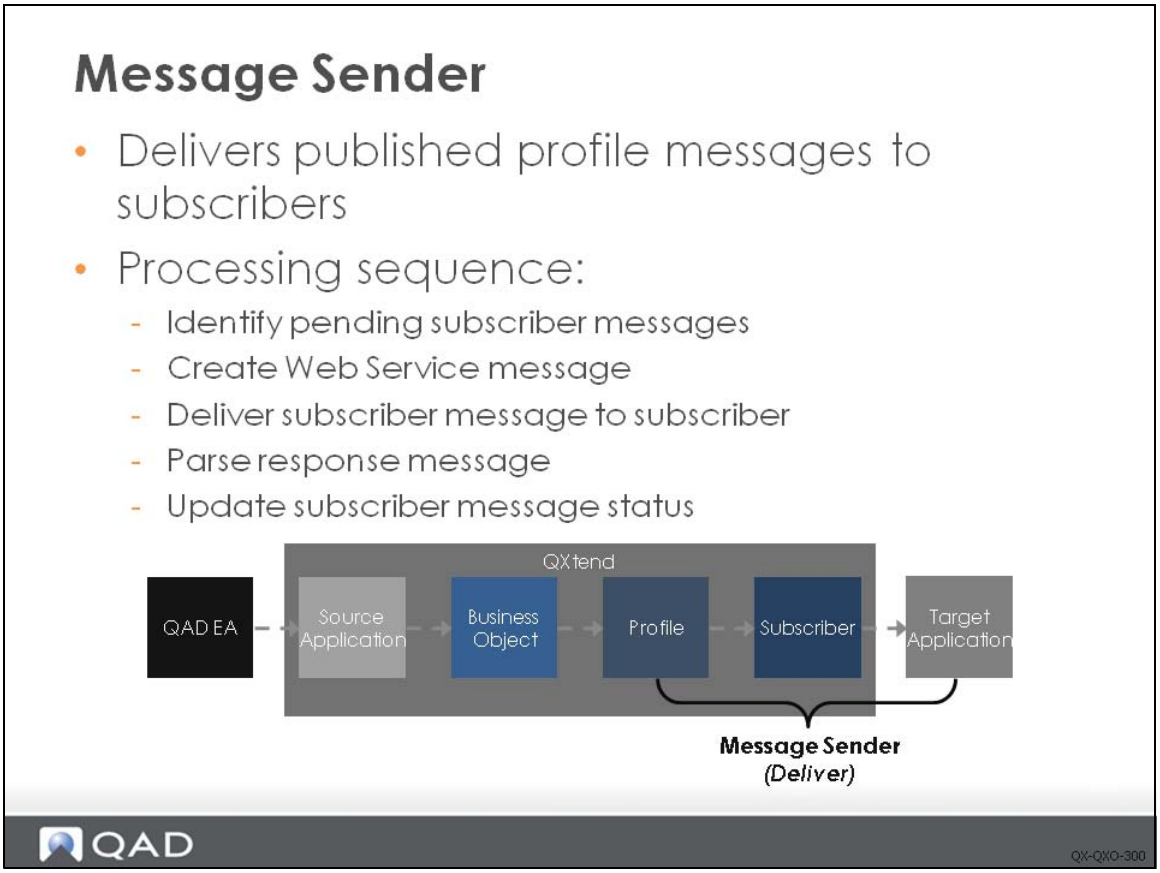
Event Types. The message publisher will create a subscriber message if the original event that created the raw message is selected. The Event Types table reflects any changes made to the event types included in a data object for a business object.

Field Substitution. The Field Substitution table displays a list of tokens and the values to substitute into those tokens when generating a profile message for the current subscriber.

- Token: Specify any required tokens to set values on the profile based on the subscriber that is receiving them.
- Fixed Value: Optionally, specify a fixed value to use in the QDoc for this field or specify a value using the format =$<node>$, in which case the field will be populated with the value in the specified node.
- Calculated Program: Optionally, enter the name of a Progress program (.p) to run. The program can be either in the QXOServer PROPATH or on the AppServer. The local version will be run if it exists; otherwise, it runs on the AppServer.

Detailed Response. Specify whether to set the suppressResponseDetail attribute of QDoc requests to true or false for a subscriber profile. If Detailed Response is set to Yes, then suppressResponseDetail is set to false.

Message Sender



The final phase of the QXO message life cycle is the deliver phase, handled by the message sender. In this phase, profile messages created in the publish phase are delivered to registered subscribers.

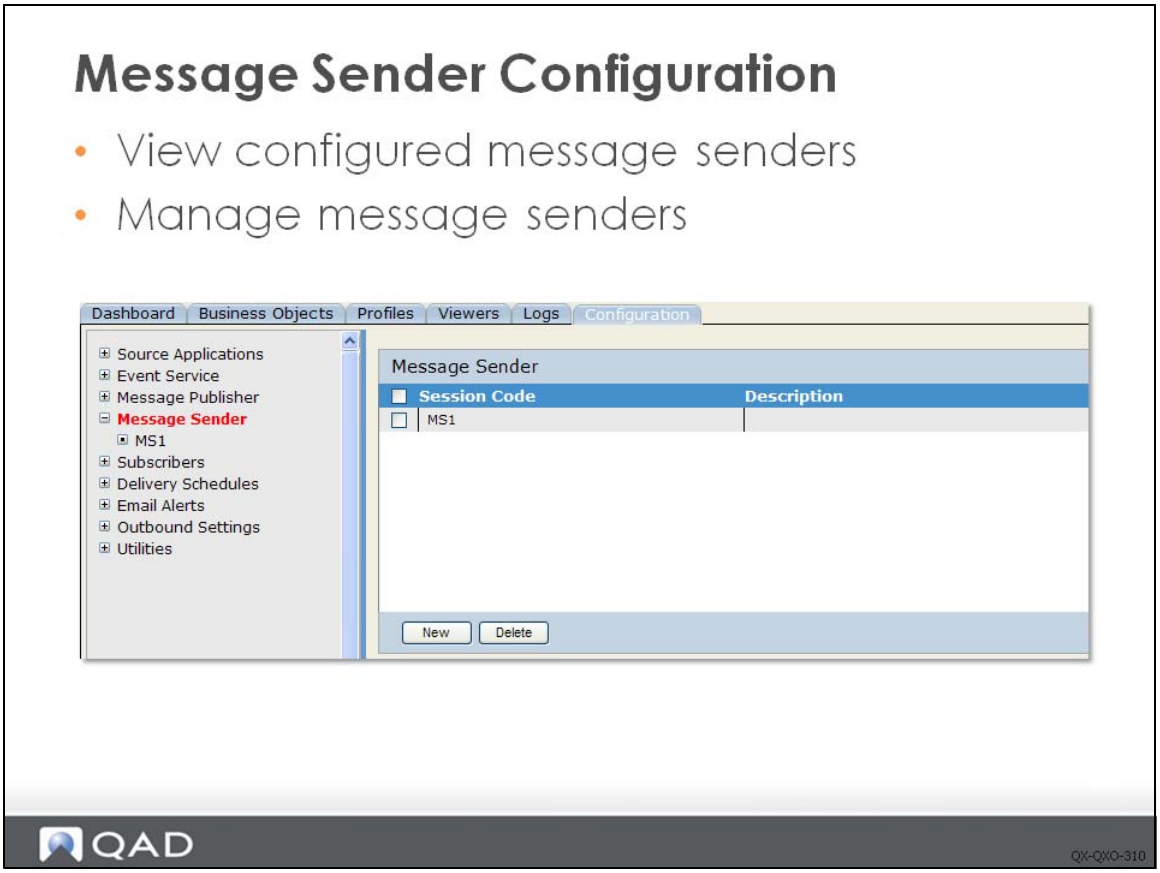
The Message Sender sends data generated by the message publisher to the registered subscribers and stores the status of the delivery in the QXO subscriber messages queue. To complete this process, several steps are required:

- Identify pending subscriber messages: Publishing creates messages in the subscriber message queue that are ready for delivery to the subscriber; these are created with a pending status. The message sender identifies any pending messages in this queue intended for subscribers that are registered with the message sender process instance; from this list the first pending message is identified. The message sender configuration allows multiple message senders to be created. Each message sender can be assigned specific subscribers, which allows advanced, flexible configuration to optimize the delivery of messages to specific subscribers by creating dedicated message senders.
- Create Web Service message: The pending subscriber message identified in the previous step (which contains only application data at this point) is now converted into a Web Service message if the Include Soap Envelope option is selected on the subscriber to which the message is being delivered. The application data is inserted into a SOAP envelope, and the SOAP envelope is updated with the values defined in the subscriber configuration.
- Deliver subscriber message: The message created in the previous step is now delivered to the subscriber using the details configured for the subscriber. The subscriber determines the delivery method and destination (file drop, generic Web Service, or QXtend Web Service).

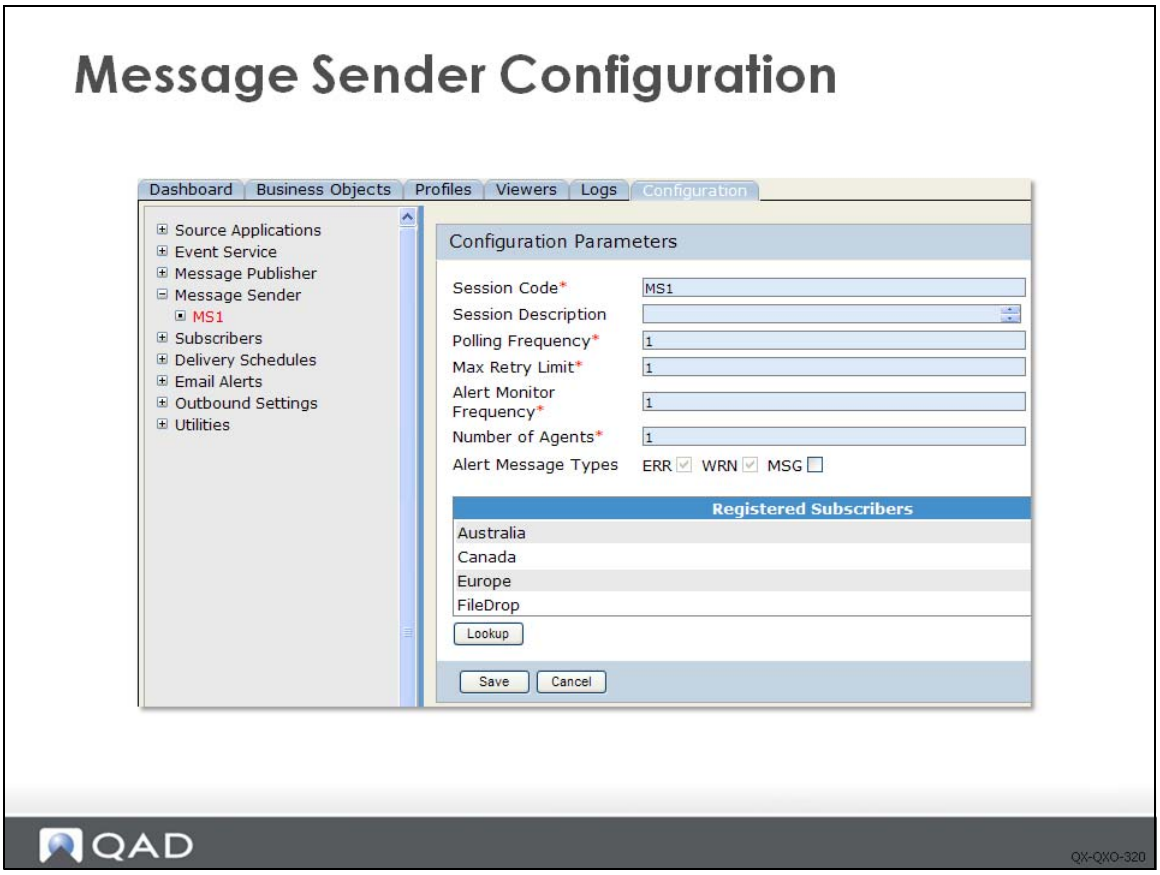
- Parse response message: When the delivery method of the subscriber is set to Web Service or QXtend Web Service, you can configure QXtend to read the response message and determine whether or not the target application processed the message successfully. However, if the delivery method is File Directory Service, this is not possible as an immediate response is not returned.

  The response message is passed to a response parser program that reads the response file, interprets the message, and extracts the processing status and any returned messages. QXO is shipped with one response parser that can read response messages returned from QXI, but you can develop custom response parsers for other message types if required. The response parser is configured in the Profile configuration screen by selecting a profile type for the profile. QXO is shipped with two types: Data Synchronization and Other. The Data Synchronization type can parse QXI response messages.
- Update subscriber message status: Based on the results of the previous step, the status of the subscriber message is updated. If there is no response parser for the response, this is typically set to DLV regardless of the contents of the response message unless the sending of the message fails due to communication or file permission problems. When a response parser is available, the status is determined based on the content of the response message.

Configuring Message Senders



Message senders are managed using the Configuration tab. You can view existing message senders by selecting the Message Sender node in the tree menu. Use the Add and Delete buttons to create and delete message senders. To modify a message sender, select the service name from the tree menu.



Use the Configuration Parameters screen to configure message senders. Complete the following fields:

Session Code. Enter a code (maximum 10 characters) for this service. Using a naming convention such as MS<*serviceName*> helps to identify the service as an event service.

Session Description. Enter a description (maximum 15 characters).

Polling Frequency. Enter the number of seconds the service should wait between polls to the pending subscriber messages queue in the `qxodb`. The `qxo` database is polled to pick up any pending subscriber messages. The time delay specified here allows the message sender to sleep before polling for pending subscriber messages, and prevents the message sender from consuming valuable CPU resources.

Max Retry Limit. Enter the maximum number of times during a single session that the service will attempt to reprocess a failed subscriber message (maximum 9999).

Alert Monitor Frequency. Specify the number of minutes between alert messages when the Alert Message Type is set to MSG and thresholds have been set.

Number of Agents. Specify the number of agents to start for the service when the service is started. Starting multiple agents increases the volume of messages that can be delivered by the message sender. In high volume environments, you can start multiple instances to meet message throughput requirements.

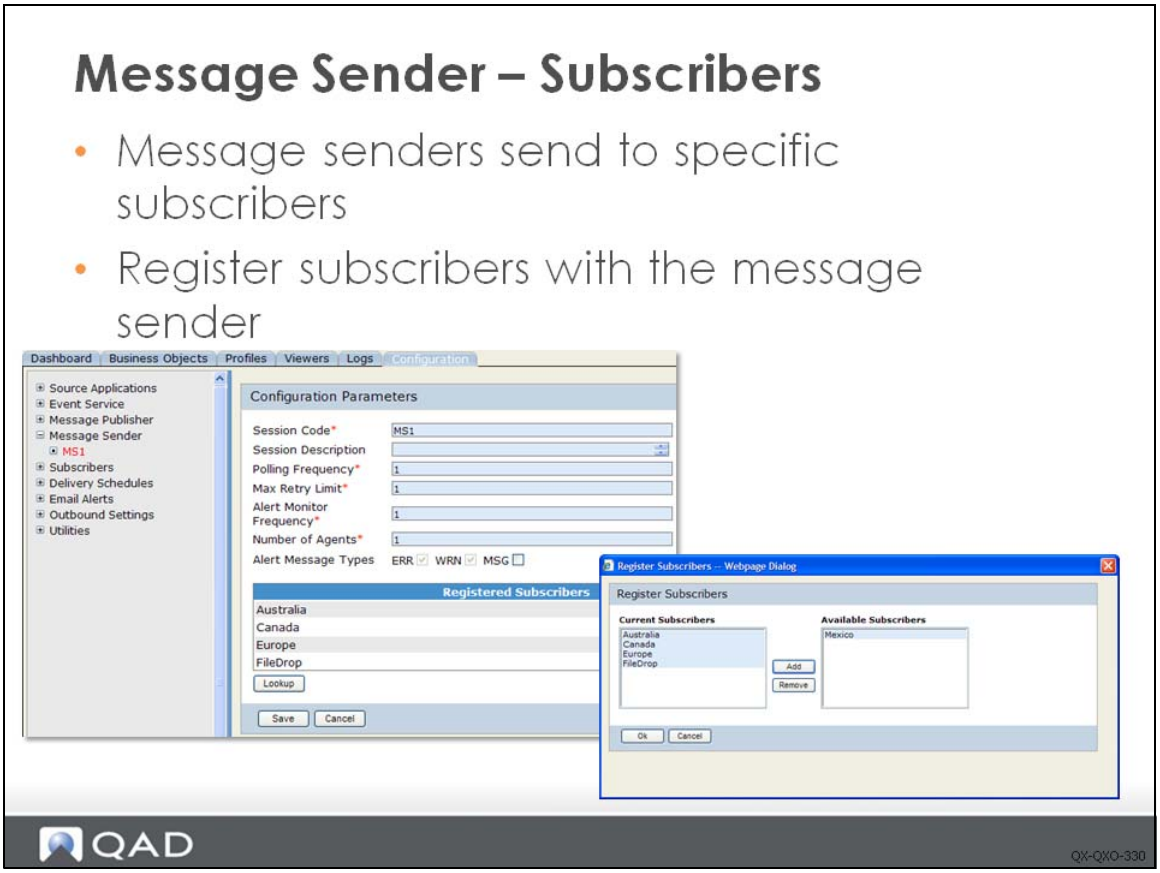
Alert Message Types. Select the message types you want to be raised as alert conditions for this profile. Select one or all of the following:

- MSG (used for metric-related alert messages)
- ERR (errors)
- WRN (warnings)

This setting works in conjunction with the email alerts functionality of QXO. Alert message types are only acted upon when email alerts are configured.

The configuration of threshold alerts for the message sender is the same as the event service and message publisher. However, threshold values apply to the number of subscriber messages.

Subscribers



Message senders deliver messages to a defined set of subscribers registered with it; this allows for flexible configuration during the implementation of QXtend. The message sender sends messages to the subscribers that are registered with it. This is useful when subscribers require minimum delay in message delivery and a dedicated message sender for a subscriber will ensure that messages are delivered quickly. Less critical subscribers can be handled by a separate message sender.

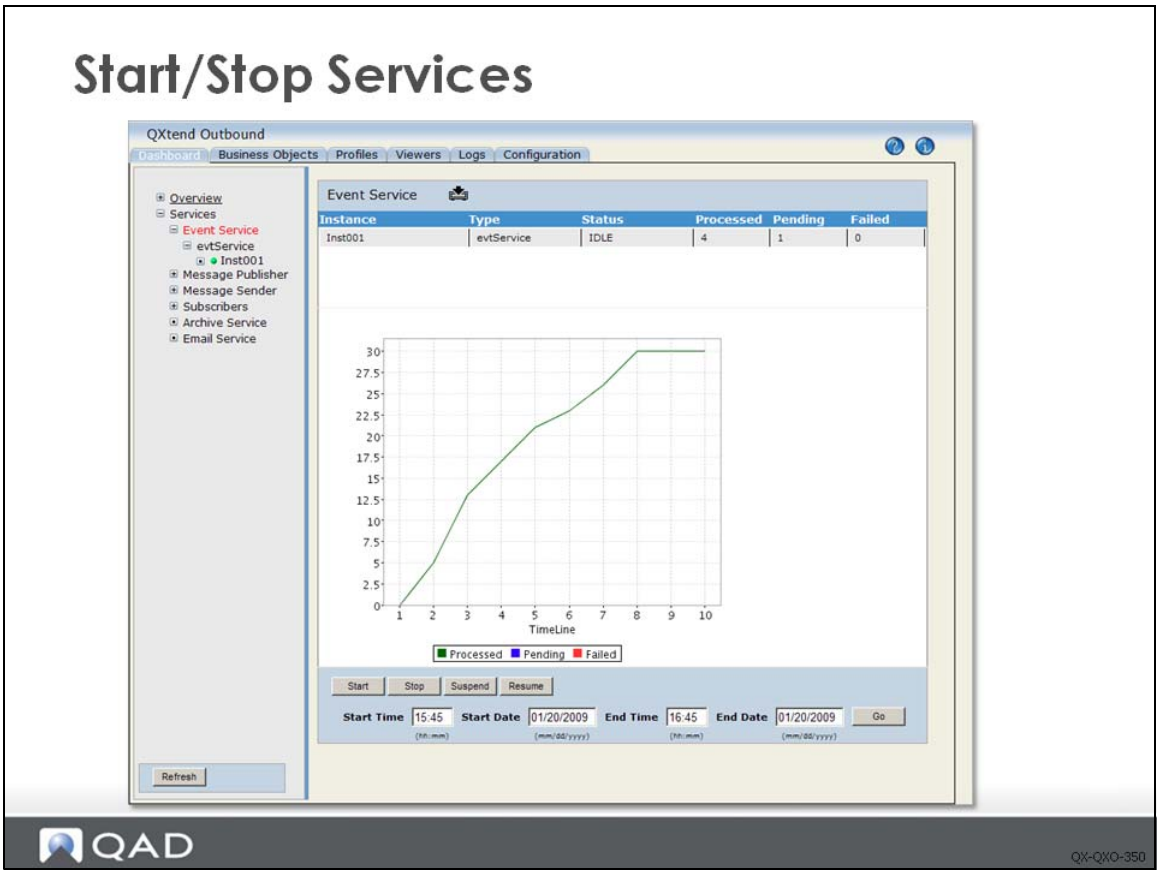
Click Lookup to display the selection box. Select the App Type, enter any filter for the Business Object search, and then click the arrow next to the Business Object Filter to execute a search and display the business object list. Select the business objects, then click Add to register them with the messages publisher.

Viewing and Controlling Services



You can view services by using the QXO Dashboard. Select the Dashboard tab, and then select the Services node in the tree menu to display a summary of service types and a visual summary of their operation. You can use the Start All, Stop All, Suspend All, and Resume All buttons to control your services as required:

- Start All: Start all services if not already started.
- Stop All: Stop all services if not already stopped.
- Suspend All: Suspend all services if not already suspended.
- Resume All: Resume all services that have been suspended.



You can view the status of an individual service by selecting the name of the service type in the tree menu. The slide here shows the status of the Inst001 instance of the event service named evtService. The table at the top of the screen shows data for this service: the instance name, type, status, and the number of messages processed, pending, and failed.

Use the Start, Stop, Suspend, and Resume buttons to control the current instance of the event service.

You also can start or stop QXO services—for example, either all services or a particular service such as the Event Service or the Message Publisher— from the command line by using the session control tool. Using this tool you can shut services down cleanly as part of a scheduled backup process, and then start them again once the qxodb server is running.

The sess-control.sh file is located in the scripts directory; the control.p file is located in the src directory.

To operate services, enter the following at the command line:

```
./sess-control.sh {QUERY|START|STOP|SUSPEND|RESUME} {SERVICES|ES|MP|MS|SA|SB} {service-name}
```

Where {QUERY|START|STOP|SUSPEND|RESUME} is the command you want to execute, {SERVICES|ES|MP|MS|SA|SB} specifies the service type and {service-name} is the name of the service you want to use.

For example, to start all services enter the following:

```
./sess-control.sh START
```

To stop all services enter the following:

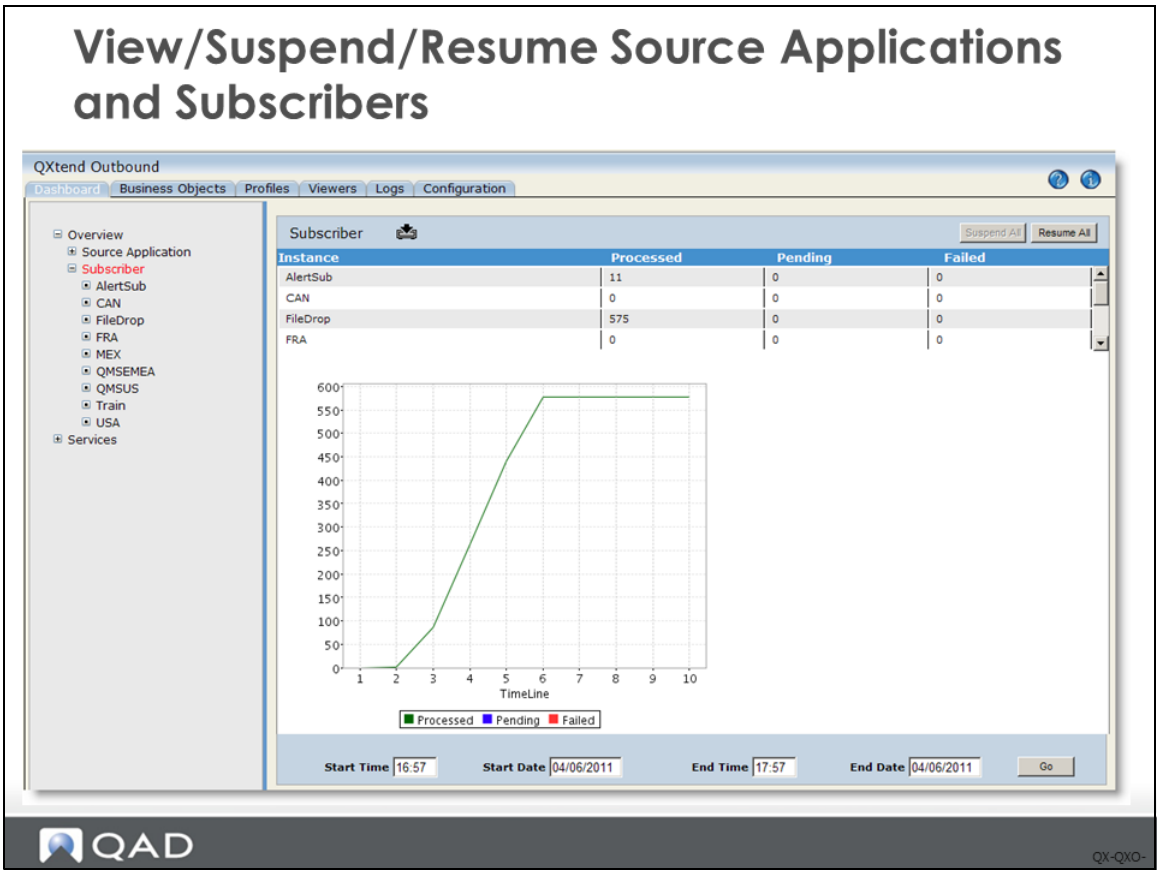
```
./sess-control.sh STOP
```

For example, to start a service named ES1 you would enter the following:

```
./sess-control.sh START ES1
```

To stop the service named ES1 you would enter the following:

```
./sess-control.sh STOP ES1
```



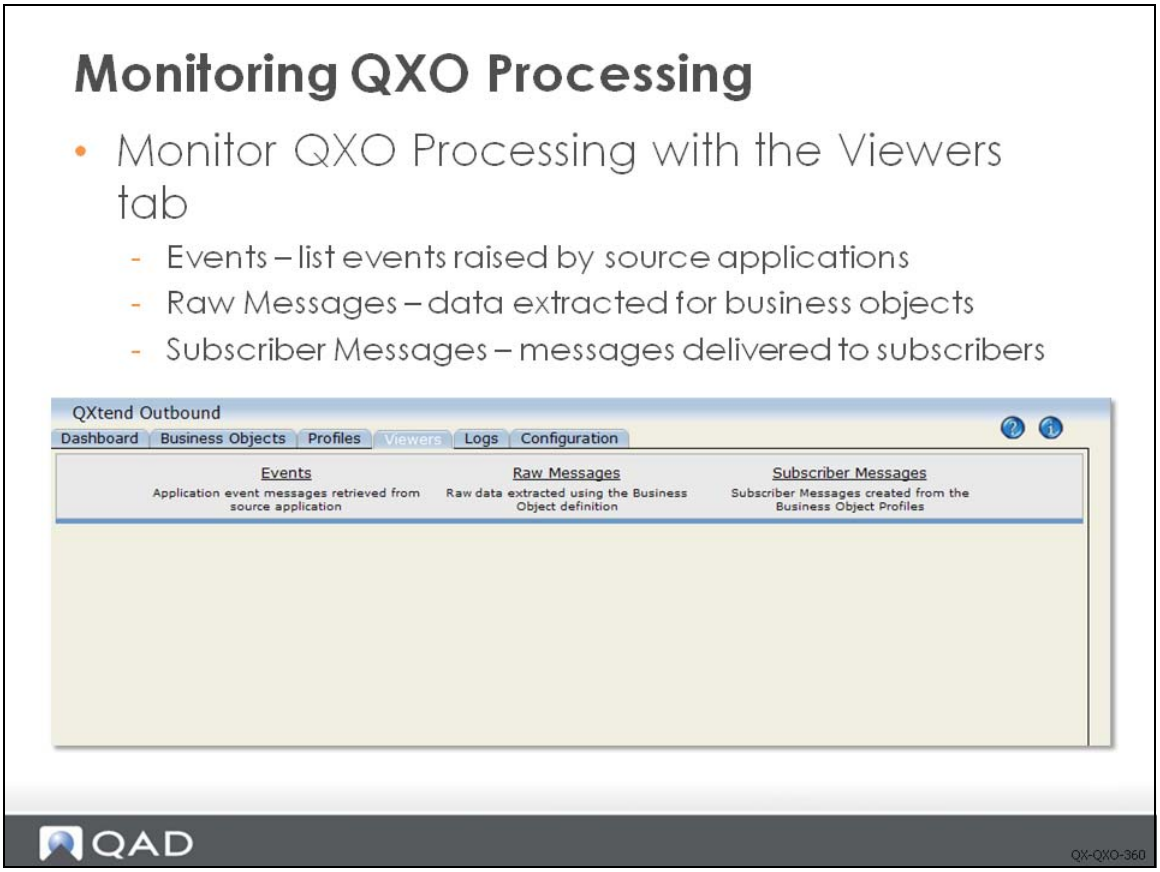
You can also monitor source applications and subscribers through the QXO Dashboard. In the Dashboard tab, select the Overview node in the tree menu and then select Source Application or Subscriber nodes to display a summary of source applications or subscribers and a visual summary of their operation. You can use the Suspend All or Resume All buttons to control them as required.

If a source application is suspended, events or messages of this source application will not be processed by any QXO services. When the source application is resumed from the suspended status, its events and messages will be processed by QXO services automatically (without restart of the services).

If a subscriber is suspended, Message Sender won? send messages to this subscriber. When the subscriber is resumed, Message Sender will resume the sending of messages to it.

From the QXO Dashboard you can also click on an individual source application or subscriber and view/suspend/resume it.

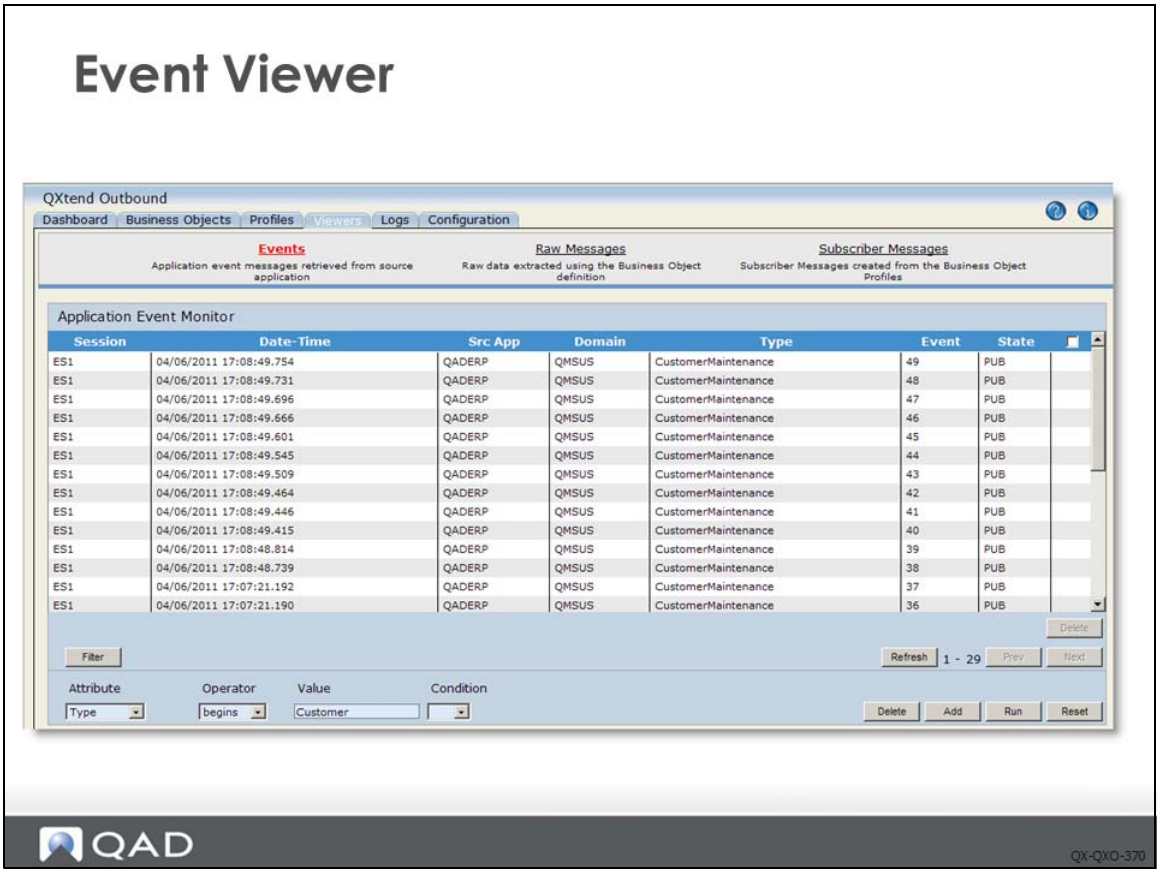
Monitoring QXO Processing



You can monitor the three phases of the QXO message life cycle by using the three screens available from the Viewers tab:

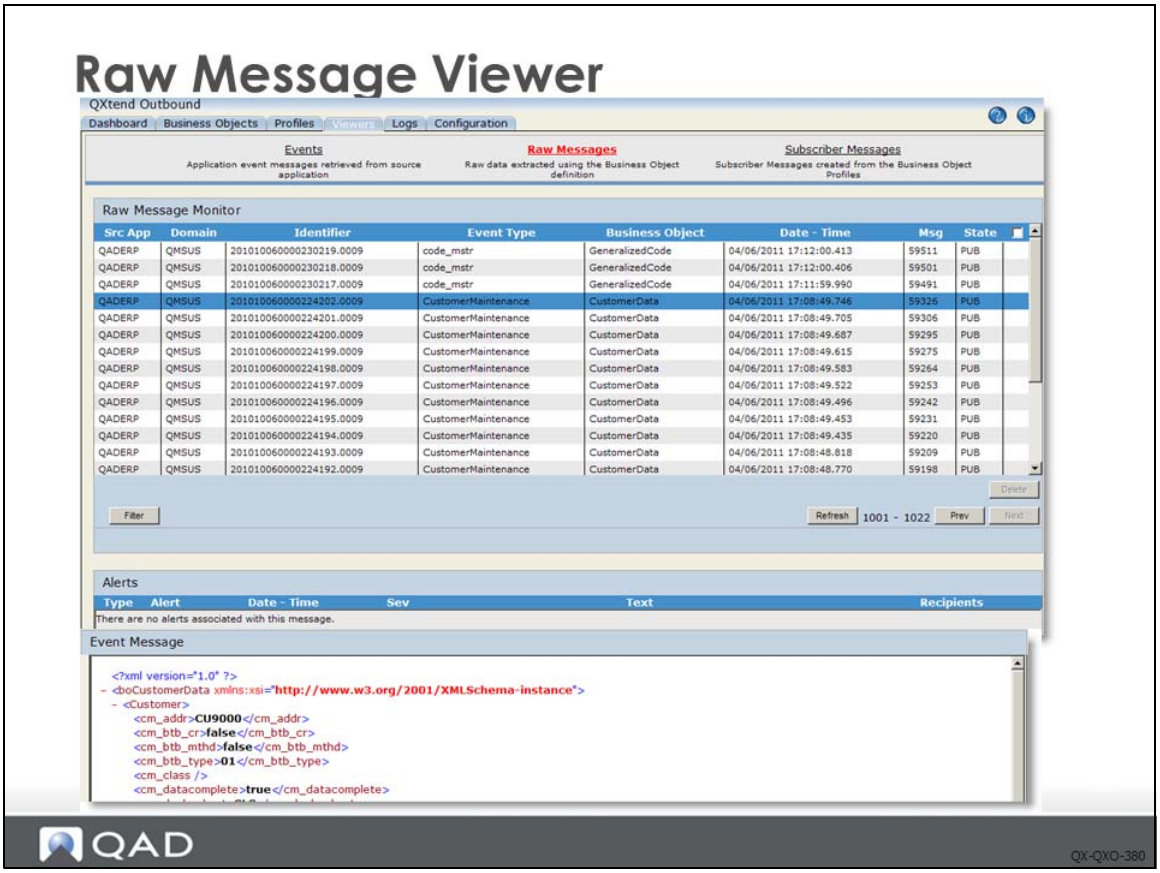
- Events: Shows all events raised by the source applications being monitored by QXO.
- Raw Messages: Shows all event messages extracted by QXO. This view displays the raw application data extracted, shown as XML.
- Subscriber Messages: Shows all subscriber messages and their processing status. You can view the request data sent to the subscriber and any response message that is returned.

Event Viewer



The Event Viewer displays details of all events raised by source applications being monitored by QXO. You can see the event service that processed the event, the source application, the domain that raised the event, the event type and the status.

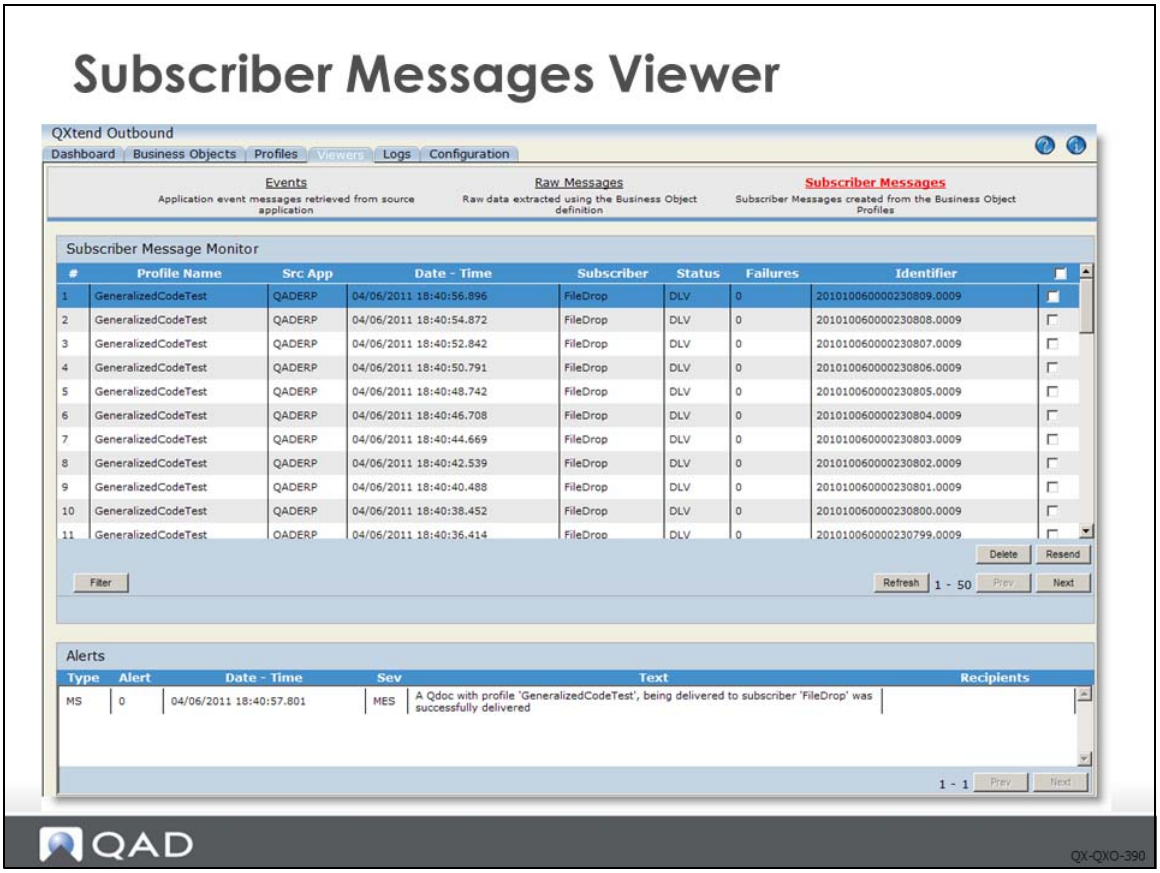
Raw Messages Viewer



The Raw Messages viewer allows you to see the event data extracted from the source application. The source application and domain are shown, as well as the unique identifier for the business object instance.

The state of the raw message indicates the current state within the publishing process. Pending messages have an empty string state value, which is used by the message publisher to identify the messages that need publishing. When the messages have been published, their state is set to PUB.

Subscriber Messages Viewer



The Subscriber Messages viewer shows all the messages for each subscriber configured within QXO, along with the message status:

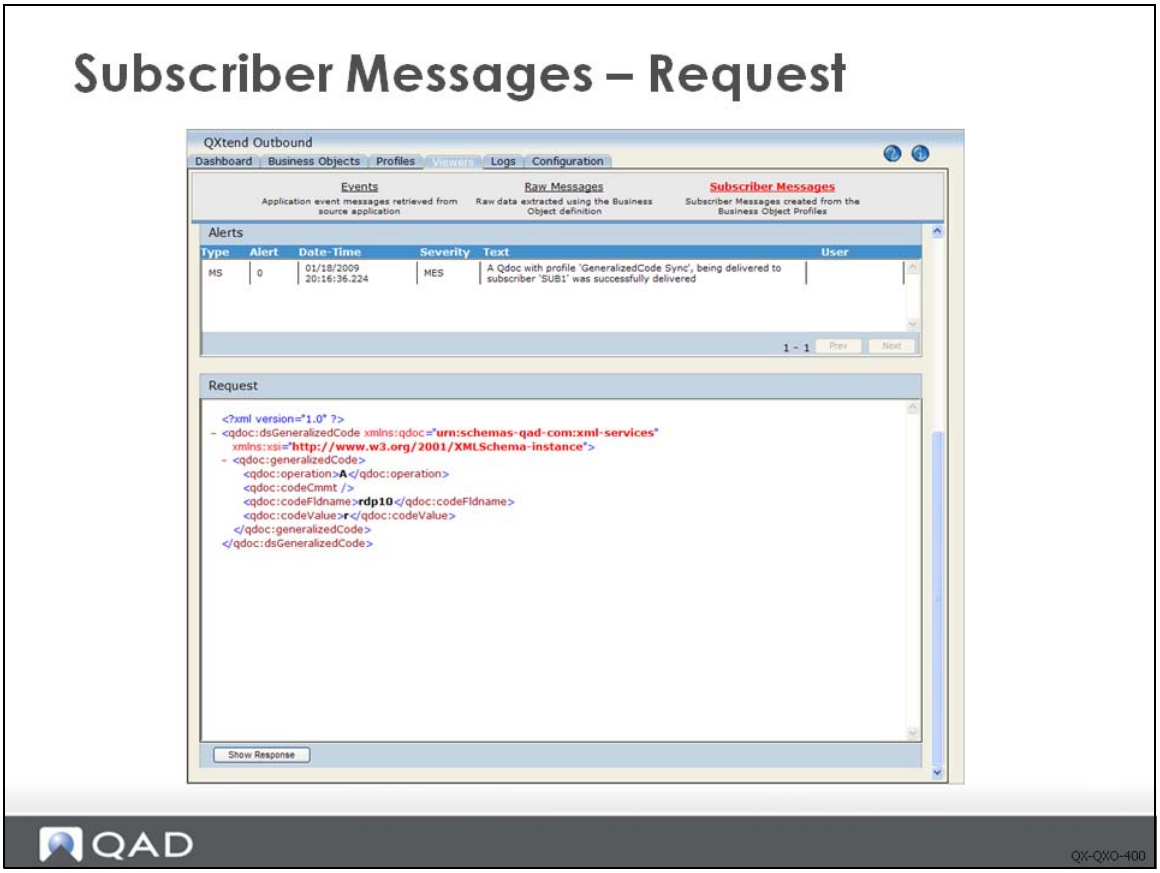
- PEND: Indicates the message has not yet been delivered to the subscriber by a message sender instance.
- SOAPERR: SOAP errors are returned from the QXtend Web Service to indicate there is a problem with either the configuration of QXI or the data submitted—for example, the receiver is invalid. A SOAPERR status indicates that the request failed the initial validation stage in QXI, and that the message was rejected before being passed to the target application.
- SENDERR: Indicates that QXO could not deliver the message to the target subscriber using the current subscriber configuration. This could be caused by an invalid URL for a web service, the user does not have permission to create files in the specified directory on a file drop subscriber. Anything that prevents QXO from delivering the message will result in a SENDERR status.
- APPERR: Indicates the message was passed to the application for processing, but the business logic of the application raised an error during processing. These types of errors are errors you would expect to see if a user enters data into an application. For example, if you supply a customer number that does not exist, an APPERR error is returned.
- WRN: Indicates that the message was successfully delivered to the subscriber; however, during the processing of the request, Warning messages were raised and require investigation.

- DLV: Indicates the message was successfully delivered to the subscriber. Note that if the profile does not have a response parser associated to it, all messages are set to DLV since it is the response parser's responsibility to read the response that is received from the target application to determine the result processing and the status to assign to the subscriber message.
- SUP: Indicates a subsequent message for this particular message instance was received before this message was delivered to the subscriber, and the Allow Superseded option on the subscriber was enabled. Effectively, this message was not delivered to the subscriber because it was superseded by a later message.
- HOLD: Indicates messages that are on HOLD have been prevented from being sent to the subscriber because an earlier message for the same data instance returned an error during the delivery phase, and the Allow Superseded option on the subscriber was not enabled. When the Allow Superseded option is not enabled, QXO deliver all messages for a particular data instance in the order in which those messages are created in QXO. If a message fails, QXtend puts current and subsequent messages on HOLD until the previous message is corrected and processed successfully.

You can use the Delete button to delete a message record.

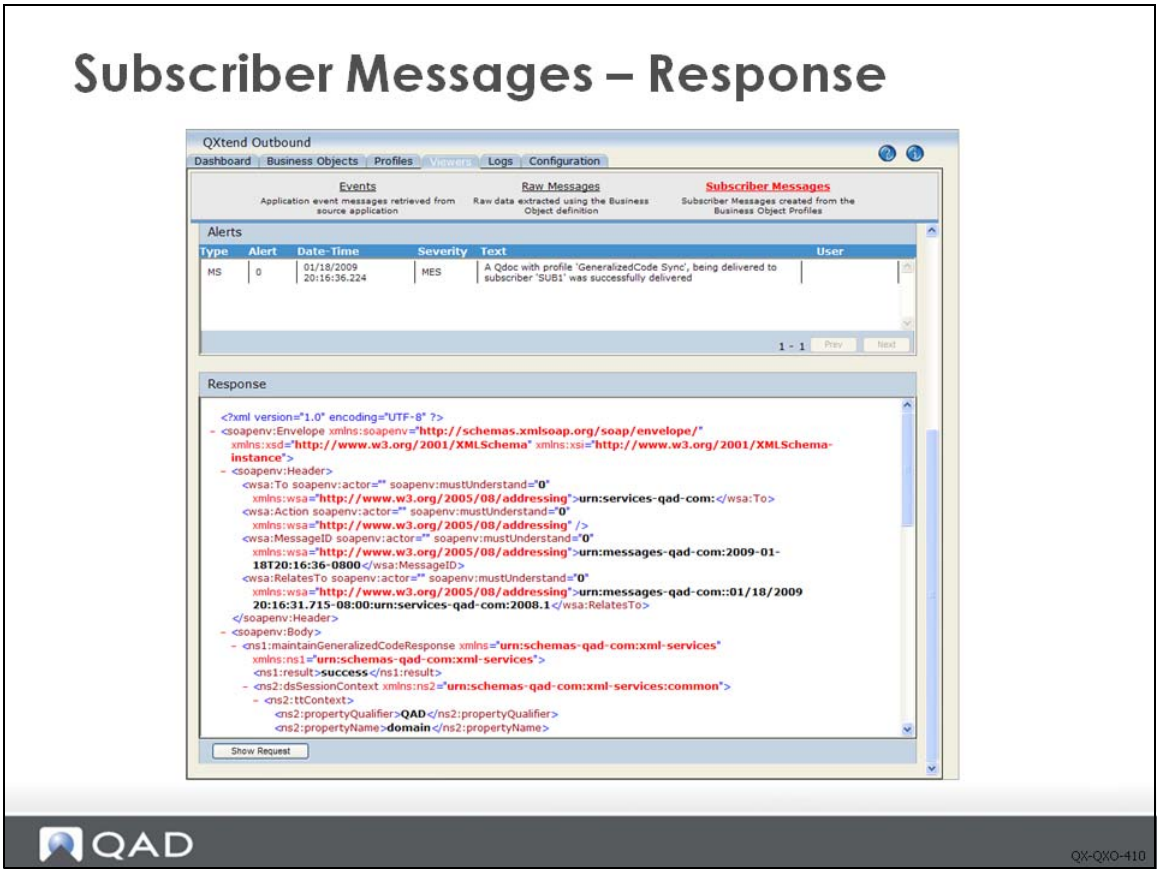
You can use Resend button to resend a message to the corresponding subscriber, no matter whether the message has been successfully sent. A resend changes the status to PEND. After delivery, the status changes to ERR or OK depending on the success of the delivery.

Subscriber Messages - Request



For each of the Subscriber Messages you can view the request sent to the subscriber and the response that is returned. By default the request is displayed. To view the response, click Show Response.

Subscriber Messages - Response



When the viewer displays the response, it shows the entire message returned by the subscriber. If the subscriber was QXI, the `result` tag contains the overall status of the request processing. To view the request again, click Show Request.

Lab: QXO Configuration

QAD QXtend 1.7.2 has been installed. Both QXI and QXO are available but neither has yet been configured. In this lab, we will configure QXO and process some test transactions to validate its installation and configuration.

QAD QXtend 1.7.2 has been installed with both QXI and QXO components. In the previous labs you performed the basic configuration required to enable requests to be processed by QXI. In the following lab, you will configure QXO to process data changes within the QAD Enterprise Applications. You also will configure the data synchronization of master data from a source domain to a set of target domains.

`<LabHomeDirectory> = C:\QXtendTraining\Labs\03-QXO-Config\`

The QXO installation installs the application but does not apply any automatic default configuration. The training describes how to configure QXO to process events raised by QAD Enterprise Applications.

1. License Manager Configuration

QAD QXtend uses a License Manager to control and track QAD QXtend usage. The License Manager component is hosted with QXI, and QXO must be told where to find the QXI that hosts the License Manager it is using. The first step in configuring QXO is configuring this information.

1 Open the QXO Web application in Internet Explorer:

`http://qaddemo:8080/qxo`

2 Select the Configuration tab.

3 Select the Outbound Settings node in the menu tree.

4 Select the License Manager option and enter the following:

Tomcat Host	qaddemo
Tomcat Port	8080
Webapp Name	qxi

5 Click the Save button.



2. Source Application Types

QXO ships with several predefined source application types; by default these are all inactive. The first step in QXO configuration is to enable the source application types that the QXO instance is going to support. In this training, you will only need to support integration to QAD Enterprise Edition (QADEE).

Enable the QADEE source application type:

1 Open the QXO Web application in Internet Explorer:

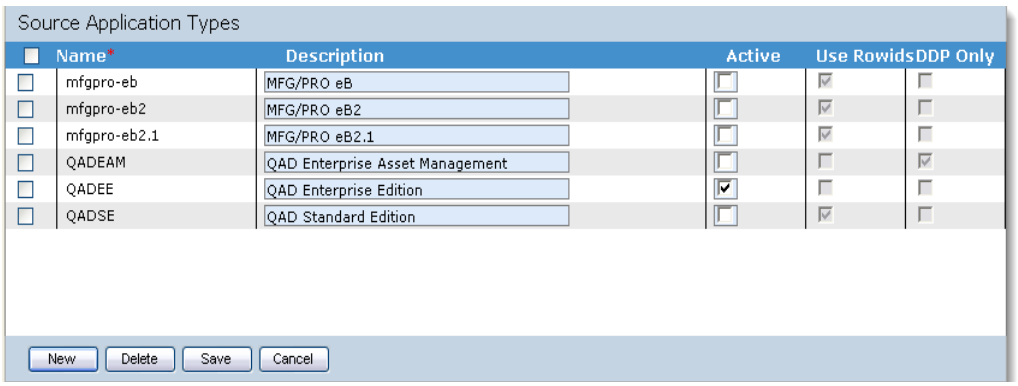
`http://qaddemo:8080/qxo`

2 Select the Configuration tab.

3 Select the Source Applications node on the menu tree.

4 Select the Active check box for the QADEE source application type.

5 Click the Save button.



3. Source Application

The source application in QAD QXtend is conceptually similar to the receiver in QXI: it identifies a QAD application instance that will act as the source of events and data. The definition of the source application includes the database set and the types of events that can be raised.

To create a new source application instance:

1 Open the QXO Web application in Internet Explorer:

`http://qaddemo:8080/qxo`

2 Select the Configuration tab.

3 Select the Source Applications node on the menu tree.

4 Select QADEE.

5 Click the New button.

6 Set the name of the source application to QADERP; the name is stored in the Code field.

7 Click the Save button.

8 Expand the Source Applications – QADEE node; the new source application instance is listed. Select the QADERP source application.

Creating the source application creates a placeholder for the source application with no configuration details. To be able to use the source application, further configuration is required.

1 Select the Databases menu option from the QADERP source application menu.

2 Create a database connection for the core QAD application database:

   a Click the New button.

   b Set the following values:

Name	/dr01/dbs/live/qaddb
Logical Name	qaddb

   c Click the Save button.

3 Create a database connection for the core QXEvents database. This database is used by source application to register events that have been raised.

   a Click the New button.

   b Set the following values:

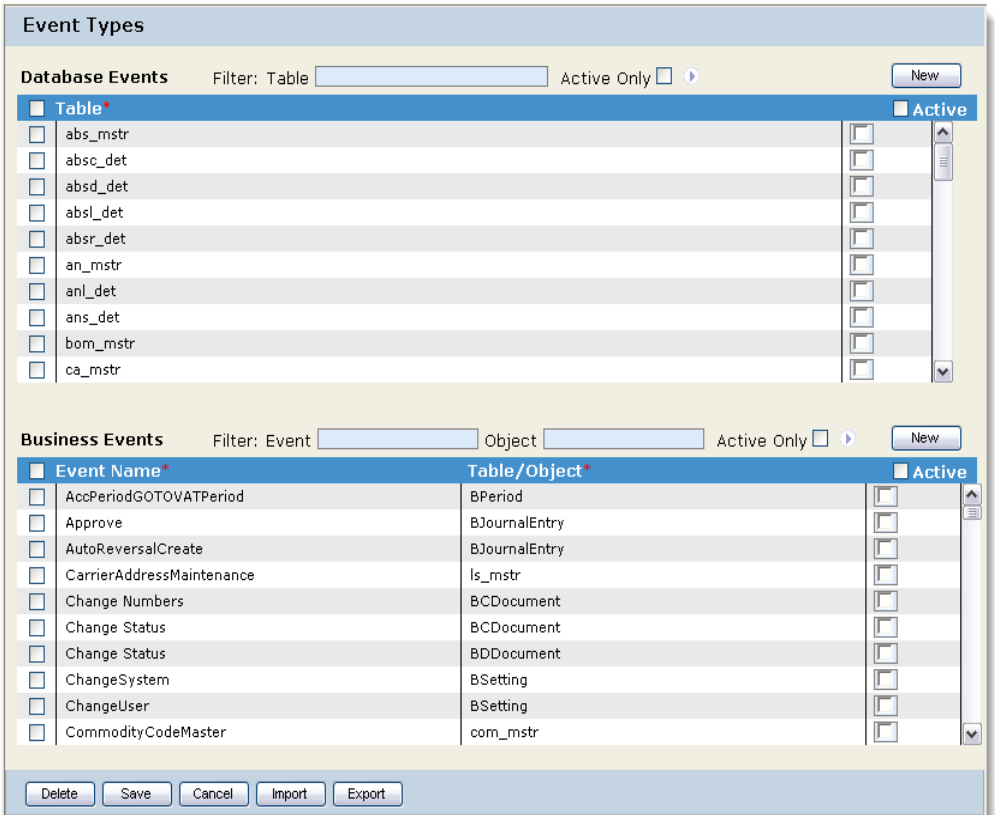
Name	/dr01/dbs/live/qxevents
Logical Name	qxevents

   c Click the Save button.

4 Select the Event Types menu option from the QADERP source application menu.

5 Click the Import button.

   Note Some business events cannot be imported if the corresponding business objects are not available. After loading default business objects and profiles, do the import again and then they should be loaded.



The source application definition also includes the details of all events that can be raised. You will configure the active events for this source application later, but for now review the list of default events. To review the events for a source application, select the Event Types node under the source application instance in the Source Application menu tree.

4. Load Default Business Objects and Profiles

QXO ships with approximately 130 predefined business objects for QADEE. Each of these has one or more default profiles provided. These business objects and profiles are not loaded as part of the standard QXO installation. The default business objects can be loaded one at a time or all at once, and are stored as XML files in the QXO installation.

The definition XML files are in the QXO Server installation directory. Review the business object and profile XML files:

1 Open PUTTY by using the icon on the training Windows machine Desktop.



2 Double-click the qaddemo saved session.

3 Log in as user `demo-admin`, password `qad`.

4 Change directory to the QXO Server installation directory

```
cd /dr01/qadapps/qea/qxtend/qxoserver
```

The `boXML/QADEE` directory contains the XML definitions for the QADEE business objects. Any subdirectories are used to hold the profile definitions for the business object.

To load all the business objects and profiles into QXO, do the following:

1. Open the QXO Web application in Internet Explorer:

   `http://qaddemo:8080/qxo`

2. Select the Configuration tab.

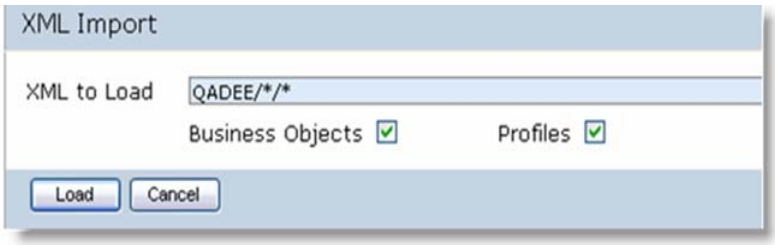
3. Select the Utilities node on the menu tree.

4. Select XML Import. The system prompts you for the XML to load in the format *<AppTypeName>/BusinessObjectName>/<ProfileName>*

5. To load all for QADEE, enter:

   `QADEE/*/*`

6. Ensure that the Business Objects and Profiles check boxes are selected.



7. Click the Load button.

QXO loads all the business objects and profiles, and reports the success/failure of the operation.

5. Review Business Objects and Profiles

The previous steps in this lab loaded the default business objects and profiles that are shipped with QXO. They are loaded for a certain source application type. In your scenario they were loaded for the QADEE source application type.

To review the business objects loaded:

1. Open the QXO Web application in Internet Explorer:

   `http://qaddemo:8080/qxo`

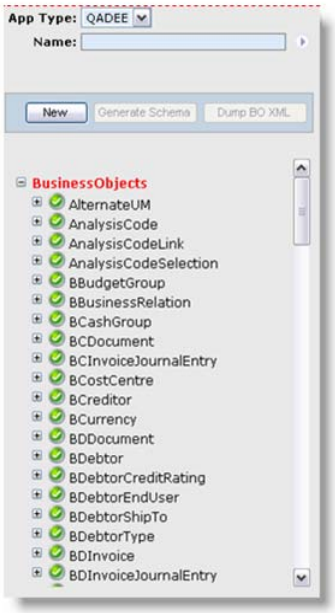
2. Select the Business Objects tab.

3. Ensure that the App Type field is set to `QADEE`.

4. To load the list of available business objects, click the arrow to the right of the Name field.



   QXO retrieves the list of available business objects and displays them as a menu tree on the left-hand side of the page.

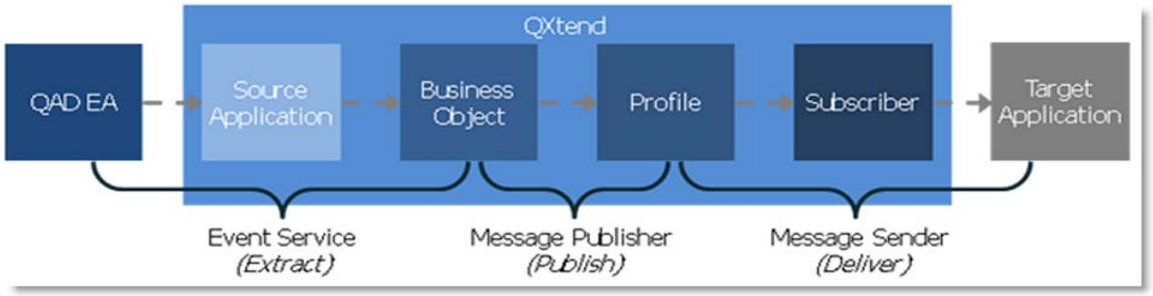


5 Review the Shipper business object. Expand each level in the business object and click each of the data objects. Review the details that display in the right-hand window.

Note The labs for the business object and profile section describe this functionality in detail.

The Profiles screen follows the same interface design. Select the Profiles tab and review some profile definitions.

6. Create QXO Services



Message processing in QXO has three phases: extract, publish, and deliver. Each phase has a service that is configured to process messages through it. The event service handles the extract, the message publisher handles the publishing, and the message sender delivers the message.

6.1 Create Event Service

The event service connects to source applications to extract data related to specific events. To create an event service:

1 Open the QXO Web application in Internet Explorer:

```
http://qaddemo:8080/qxo
```

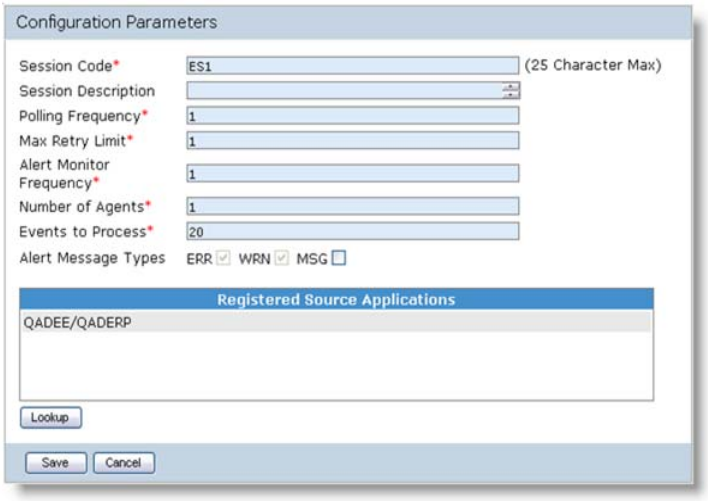
2 Select the Configuration tab.

3 Select the Event Service node on the menu tree.

4 Click the New button.

5 Create a new event service with the following settings:

- Session Code = `ES1`
- Leave all other options set to the default values.
- Register the QADERP source application with the event service. The event service instance only processes events for those source applications registered with it.



6 Click the Save button.

6.2 Create Message Publisher

The message publisher maps the data extracted from the business objects into the format defined in each of the profiles. To create a message publisher:

1 Open the QXO Web application in Internet Explorer:

`http://qaddemo:8080/qxo`

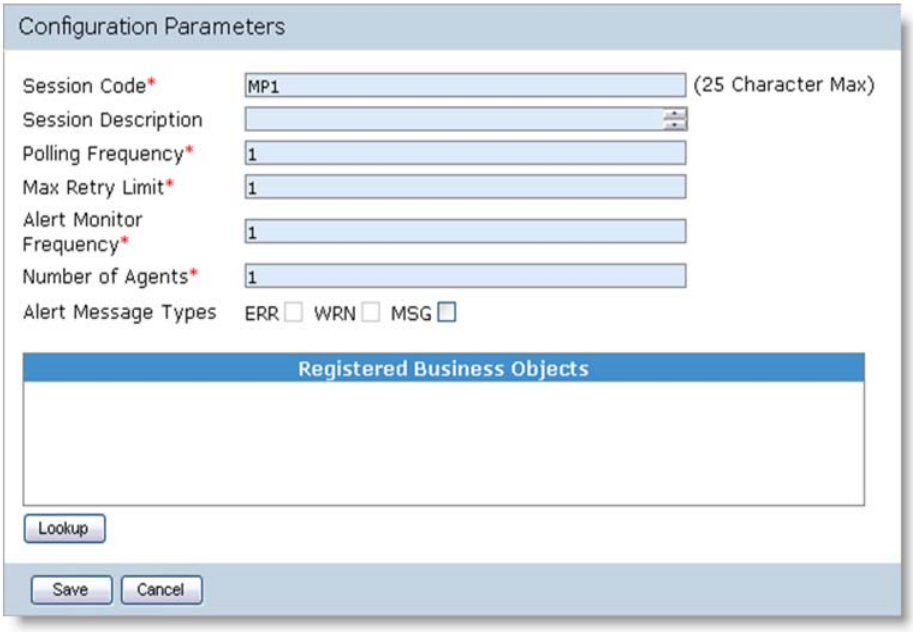
2 Select the Configuration tab.

3 Select the Message Publisher node in the menu tree.

4 Click the New button.

5 Create a new message publisher with the following settings:

- Session Code = `MP1`
- Leave all other options set to the default values.
- Do not register any business object with the publisher.



6 Click the Save button.

6.3 Create Message Sender

The message sender delivers the messages generated by the publisher to the registered subscribers. To create a message sender:

1 Open the QXO Web application in Internet Explorer:

`http://qaddemo:8080/qxo`

2 Select the Configuration tab.

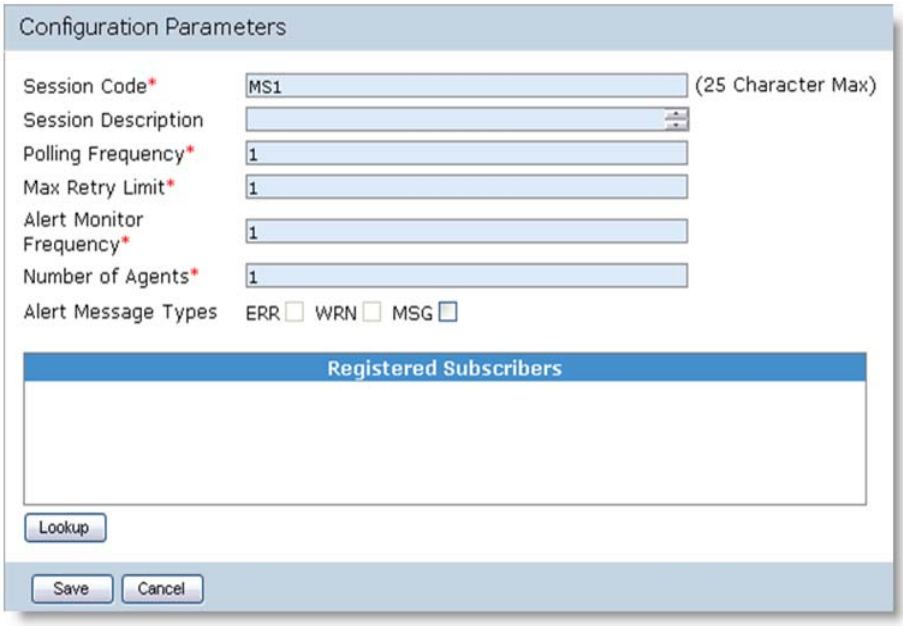
3 Select the Message Sender node on the menu tree.

4 Click the New button.

5 Create a new message sender with the following settings:

- Session Code = `MS1`
- Leave all other options set to the default values.
- Do not register any subscribers with the sender.

6 Click the Save button.

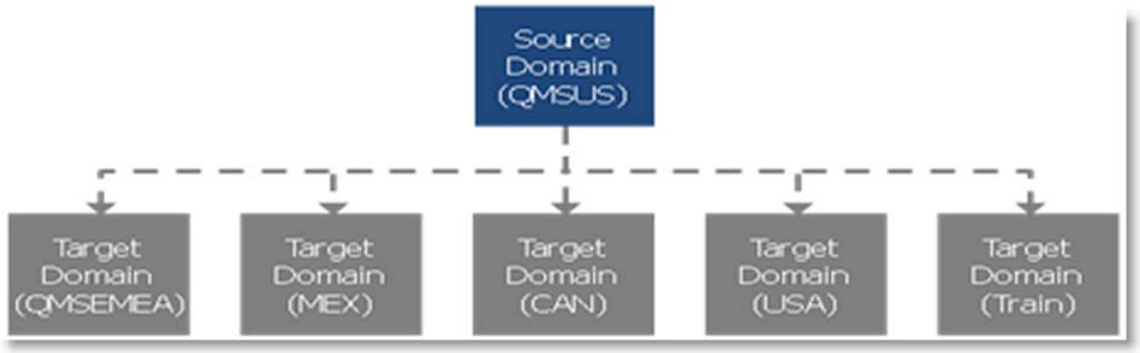


7. Configure Generalized Code Data Synchronization

The configuration described in Section 6 of this lab created the basic services required to process messages through the QXO message process flow. However, further configuration is required to start processing messages through QXO. The messages that can be processed by QXO are currently restricted because the license in QAD QXtend only supports automatic synchronization of master data between domains.

Configuring data synchronization is a straightforward task. When configuring data synchronization, you must define the domain that will maintain the master set of data. In this scenario the master domain is 10USA. The next step is to determine which domains are going to receive updates from the master domain; in this scenario the target domains are 11CAN and 12MEX.

Finally you must identify the master data objects that you want to synchronize between from the master domain to the target domains. In this training scenario we will start by synchronizing the Generalized Codes data object.



Several configuration steps are required for the scenario above:

1 Enable QXO in QAD Enterprise Edition.

2 Enable the event type in the source application.

3 Link the business objects being published to the message publisher.

4 Define subscribers for each of the target domains.

5 Link the subscribers to the message sender.

7.1 Enable QXO in QAD Enterprise Edition

When QXO is installed in the QAD EE, the publishing of events by default is disabled, meaning that business objects are not created as data changes. To enable the creation of events, you must first enable QXO.

To enable QXO:

1 Open the QAD Enterprise Applications from the Windows Desktop.



2 Log in with user id = `qmi`, password = blank.

3 Open the QXtend Outbound Control (36.16.19) function.

4 Ensure that the Enable QXtend Outbound check box is selected.

5 Save the changes.

7.2 Enable Event Types in QXO

The event types in QXO are maintained on the source application instance under the Event Types node on the menu tree. By default all events are not activated when QXO is installed. In the scenario you are implementing, you know you will monitor the Generalized Code event, so you must enable the `code_mstr` event type.

To enable the `code_mstr` event type:

1 Open the QXO Web application in Internet Explorer:

```
http://qaddemo:8080/qxo
```

2 Select the Configuration tab.

3 Select the Source Applications node on the menu tree.

4 Select QADEE.

5 Select the QADERP source application instance.

6 Select the Event Types option.

7 Locate `code_mstrco` and select the Active check box.

8 In the Data Object Listening dialog, select the Listen check box for GeneralizedCode/GeneralizedCode.

7.3 Update Message Publisher

In Section 6 you created a message publisher called MP1; this needs to be updated with the business objects that we want to publish from the source application. In this scenario we want to publish data from the Generalized Code business object. The following steps show how to configure the publisher to publish data for Generalized Codes:

1 Open the QXO Web application in Internet Explorer:

`http://qaddemo:8080/qxo`

2 Select the Configuration tab.

3 Select the Message Publisher node on the menu tree.

4 Select the MP1 message publisher instance on the menu tree. Click the Lookup button to register business objects with the publisher.

5 Ensure the App Type is set to QADEE.

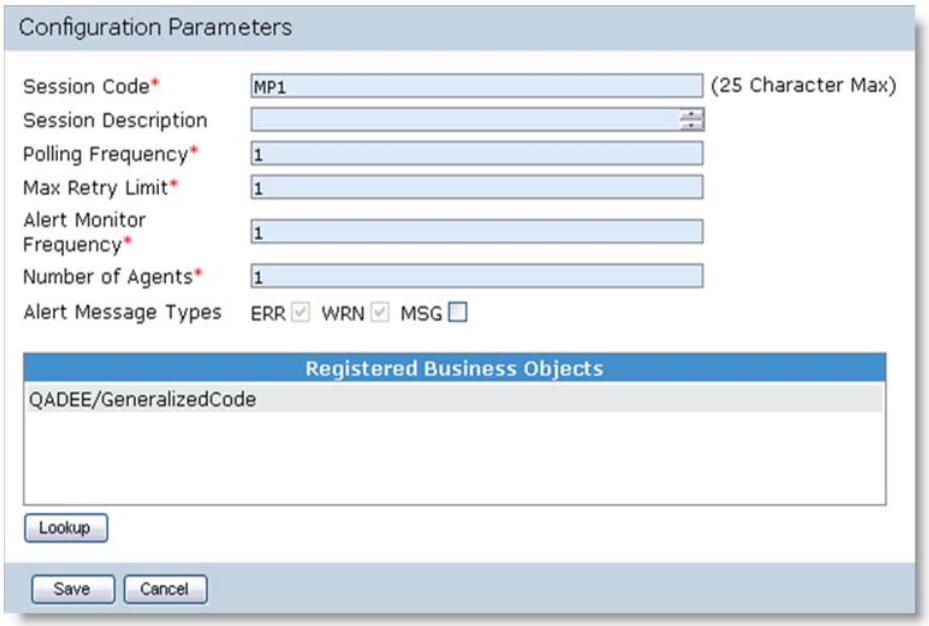
6 Click the fetch button next to the Business Object Filter prompt to display a list of available business objects.

7 Select the QADEE/GeneralizedCode business object.

8 Click the Add button.

9 Click OK to return to the publisher configuration screen.

10 Click the Save button.



7.4 Create Subscribers

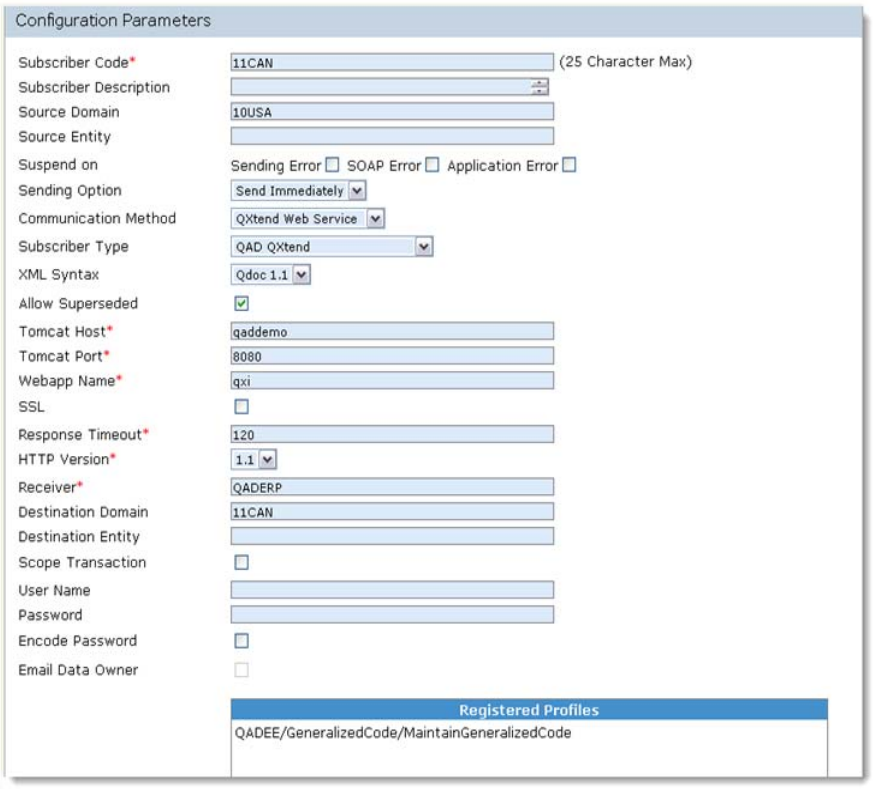
The subscriber links QXO to the target applications that data is being delivered to. In the data synchronization scenario each of the target domains is a target application, so you need to create a subscriber for each of target domain.

Follow these steps to create the subscribers:

1. Open the QXO Web application in Internet Explorer:

   `http://qaddemo:8080/qxo`

2. Select the Configuration tab.
3. Select the Subscribers node on the menu tree.
4. Click the New button.
5. To configure the subscriber, do the following:
   a. In the Subscriber Code field enter `11CAN`.
   b. In the Source Domain field enter `10USA`.
   c. Set the Communication Method to QXtend Web Service.
   d. Set the Subscriber Type to QAD QXtend.
   e. Set the XML Syntax to QDoc 1.1.
   f. Select the Allow Superseded check box.
   g. In the Tomcat Host field enter `qaddemo`.
   h. In the Tomcat Port field enter `8080`.
   i. In the Webapp Name field enter `qxi`.
   j. In the Receiver field enter `QADERP` (the receiver configured in the QXI Configuration lab).
   k. In the Destination Domain field enter `11CAN` (the domain to load data into).
   l. To register the profile messages that are sent to the subscriber, click the Register Profiles button.
   m. Ensure the App Type is set to `QADEE`.
   n. Click the fetch button next to the Filter prompt to display a list of available profiles.
   o. Select the QADEE/GeneralizedCode/MaintainGeneralizedCode profile.
   p. Click the Add button.
   q. Click OK to return to the subscriber configuration screen.
   r. Click the Save button.



6 Repeat Steps 4 and 5 for the 12MEX domain. Ensure that the destination domain is set to the target domain being processed.

You have now created all subscribers required for the scenario.

7.5 Add Subscribers to Message Sender

Now you assign the subscribers created in Step 7.4 to the message sender. A message sender instance will only deliver messages to subscribers that are registered with it.

To register subscribers with the message sender:

1 Open the QXO Web application in Internet Explorer:

`http://qaddemo:8080/qxo`

2 Select the Configuration tab.

3 Select the Message Sender node on the menu tree.

4 Select the MS1 message publisher instance in the menu tree.

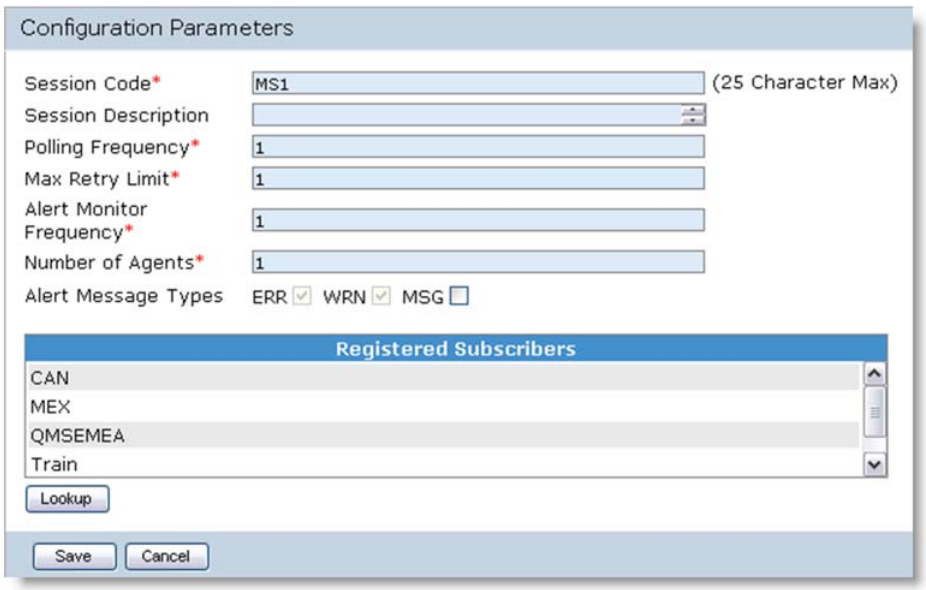
5 Click the Lookup button to register subscribers with the sender.

6 Select 11CAN and 12MEX from the Registered Subscribers list.

7 Click the Add button.

8 Click OK to return to the publisher configuration screen.

9 Click the Save button.



8. QXI Receiver Configuration

The maintainGeneralizedCode profile that the subscribers are configured to receive generates a maintainGeneralizedCode QDoc with the version eB2_2. The QADERP receiver that is configured in QXI must have this API enabled in order for messages to be successfully processed from QXO to QXI; otherwise, the messages will fail with a `QDoc not supported` error.

Using the process used in the QXI Configuration lab, add support for the maintainGeneralizedCode-eB2_2 API to the QADERP receiver.

9. Start the QXO Services

QAD QXtend Outbound is now configured to perform data synchronization of Generalized Code data from the 10USA domain to the other domains configured in QADEE. However, before you start to synchronize data between domains we need to make sure that all of the QXO services are running.

To start the QXO Services:

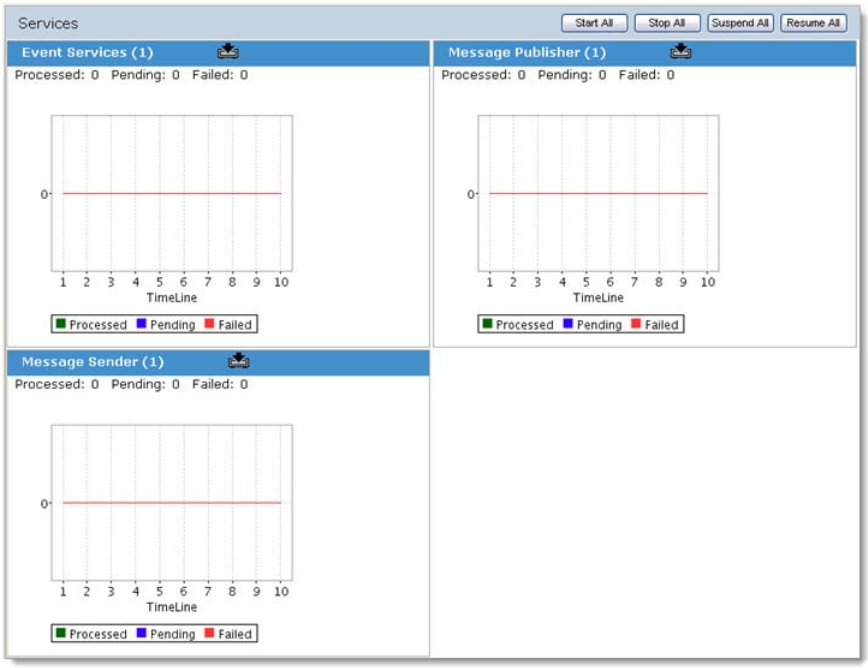
1 Open the QXO Web application in Internet Explorer:

`http://qaddemo:8080/qxo`

2 Select the Dashboard tab.

3 Select the Services node on the menu tree.

4 Click the Start All button to start the QXO services. The number shown on the graphs next to each of the services should change from 0 to 1.



10. Test the Data Synchronization Configuration

Now that we have finished configuring QXO and QXI, you can now create a new Generalized Code in the 10USA domain.

10.1 Create New Generalized Code

1. Open the QAD Enterprise Applications from the Windows Desktop.
2. Log in with user id = `qmi`, password = blank.
3. Open the Generalized Codes Maintenance function(36.2.13).
4. Check the generalized codes in the target domains (11CAN and 12MEX).
5. Create a new generalized code entry in the 10USA domain that is not present in any of the target domains.

10.2 Check QXO Viewers

Once the new Generalized Code has been created, QXO will pick up the event and move it through the message lifecycle of QXO. To validate that the message has been picked up and passed to the target domains, follow these steps:

1. Open the QXO Web application in Internet Explorer:

   `http://qaddemo:8080/qxo`

2 Select the Viewers tab.

3 Select the Subscriber Messages view.

4 You should see five Generalized Code Sync messages that have been delivered to the target domains.

5 In the Subscriber Messages view, examine the details of the request message sent to each subscriber. Also examine the response message that is returned.

10.3 Check Target Domains

Verify the new Generalized Code has been populated into each of the target domains.

10.4 Data Synchronization Delete

1 Delete the Generalized code that was created in 9.2.

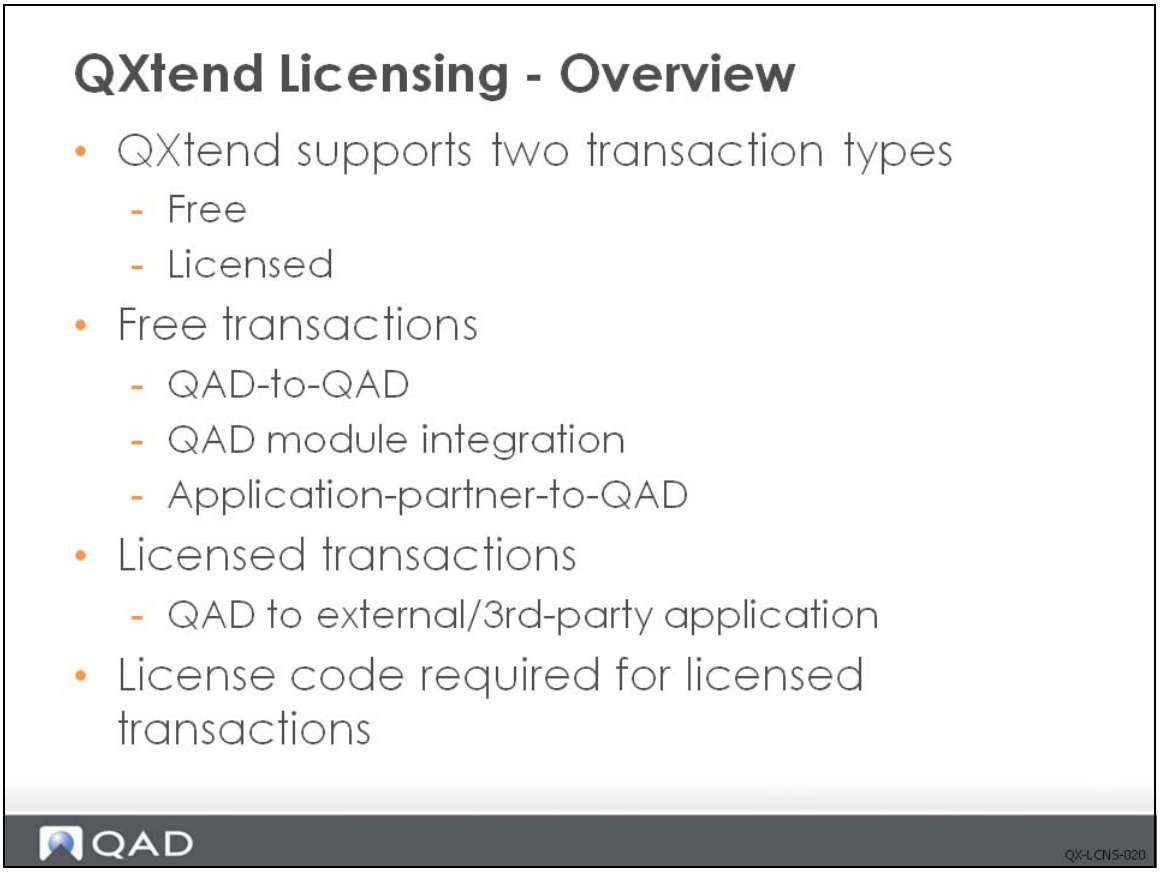
2 Verify the messages were generated and processed in QXO.

3 Verify the data has been deleted from the target domains.

Chapter 4

Licensing

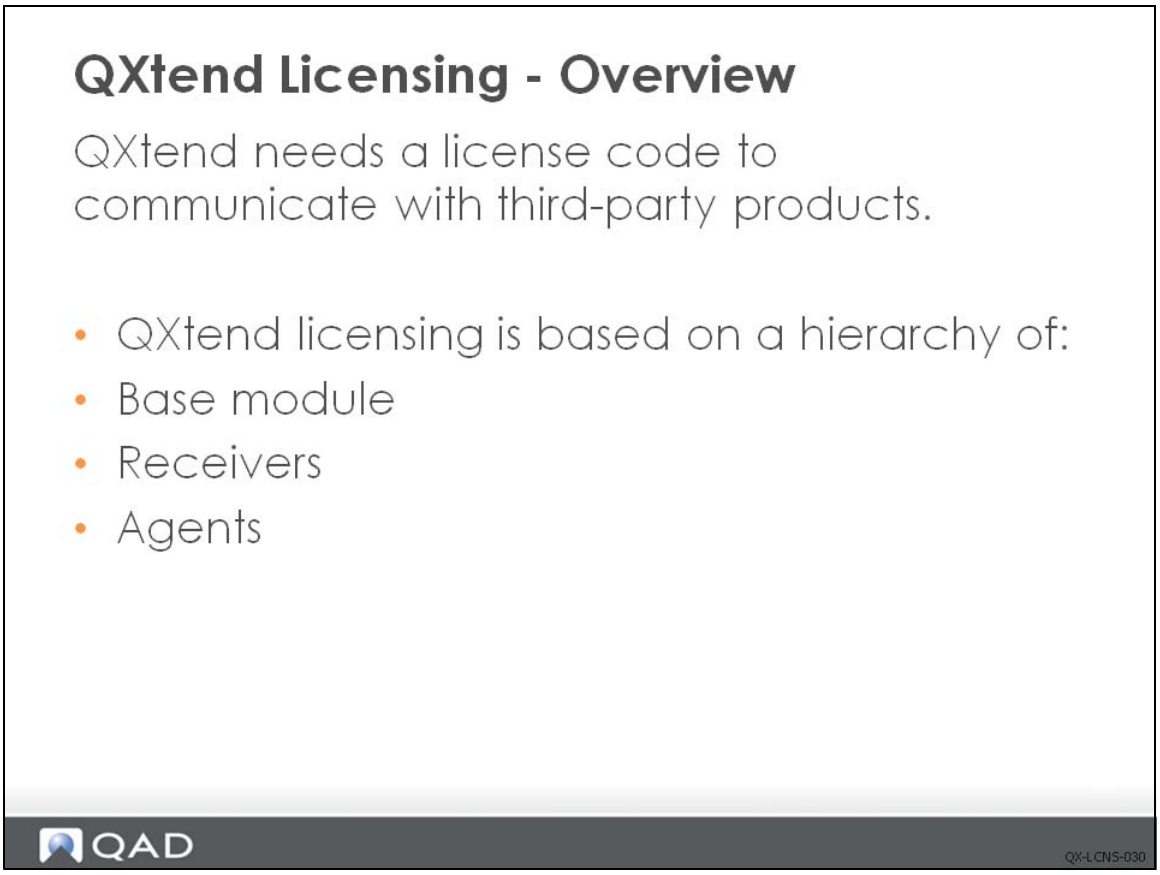
Overview



QAD QXtend is an integration product that supports both the integration of internal QAD modules, and the integration of QAD applications to legacy systems and third-party applications. QXtend essentially aims to provide an integration backbone across the entire QAD Enterprise Applications suite that is used universally for integration.

However, when QXtend is used to integrate with external third-party applications, it simplifies the implementation and provides a QAD-approved standard approach to integration implementation, management, and maintenance. Using QXtend adds value to the business and facilitates the automation of business processes to drive operating efficiency. Therefore QXtend supports two modes of operation:

- Free: The free mode allows QAD and QAD partners to use the capabilities of QXtend to integrate modules and applications without the need of any special license codes. The free mode license is installed as default with any QXtend installation and no license code is required to be entered after installation.
- Licensed: The licensed model requires the integration to third-party applications and requires QXtend to be licensed for this type of integration. QXtend prevents transactions from being processed with third-party applications unless a license code has been entered after the installation of QXtend.

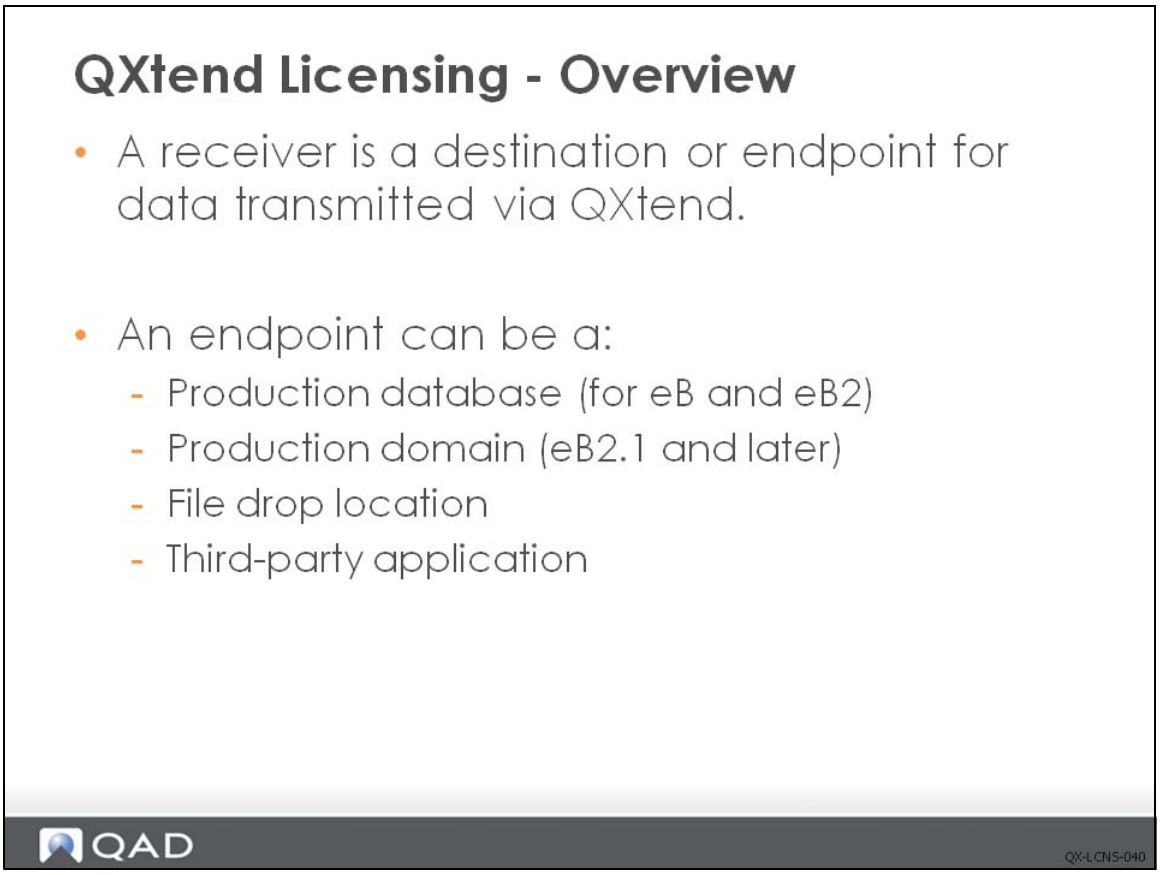


In order for QXtend to communicate with other products, you must submit a license code issued by QAD.

QXtend licensing is based on a hierarchy of the base module, receivers, and agents.

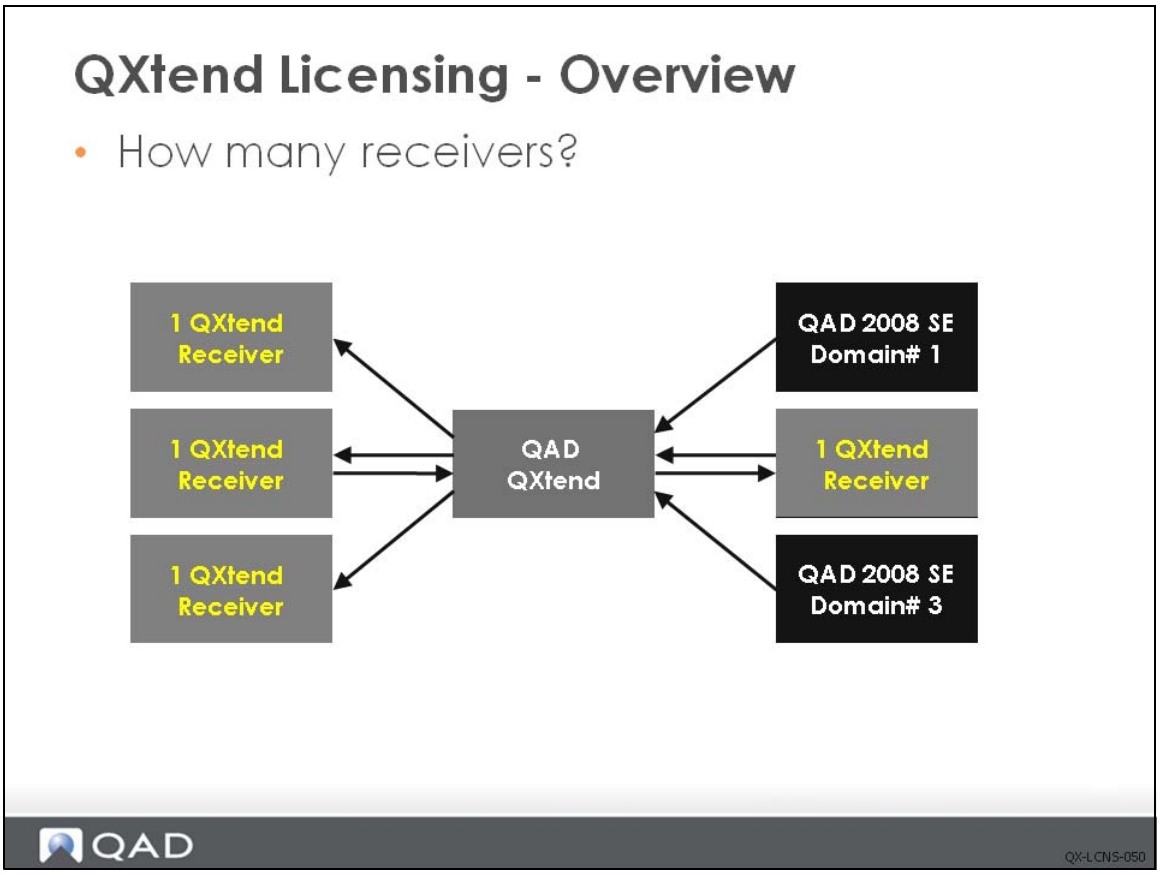
The base module requires a number of receivers and agents. Each receiver is coupled with an additional agent. You can purchase additional agents without receivers. When you purchase a receiver you get an additional agent bundled with the receiver.

The base module becomes available when you enter a valid license. The license encodes how many receivers and agents are available for each QXtend installation, the type of license, and a unique code indicating the organization who owns the license.

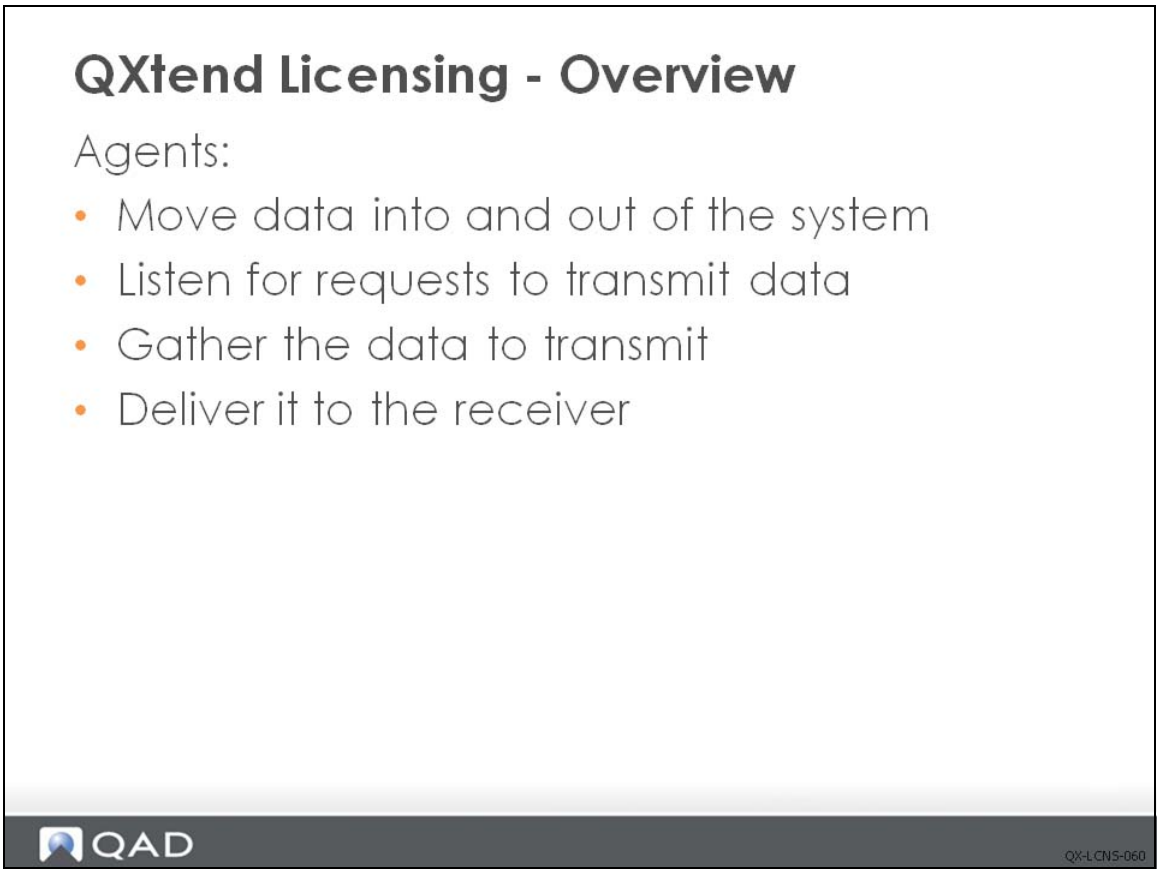


A receiver is a destination or endpoint for data transmitted via QXtend. An endpoint can be any of the following:

- Production database (for eB and eB2)
- Production domain (eB2.1 and later)
- File drop location
- Third-party application



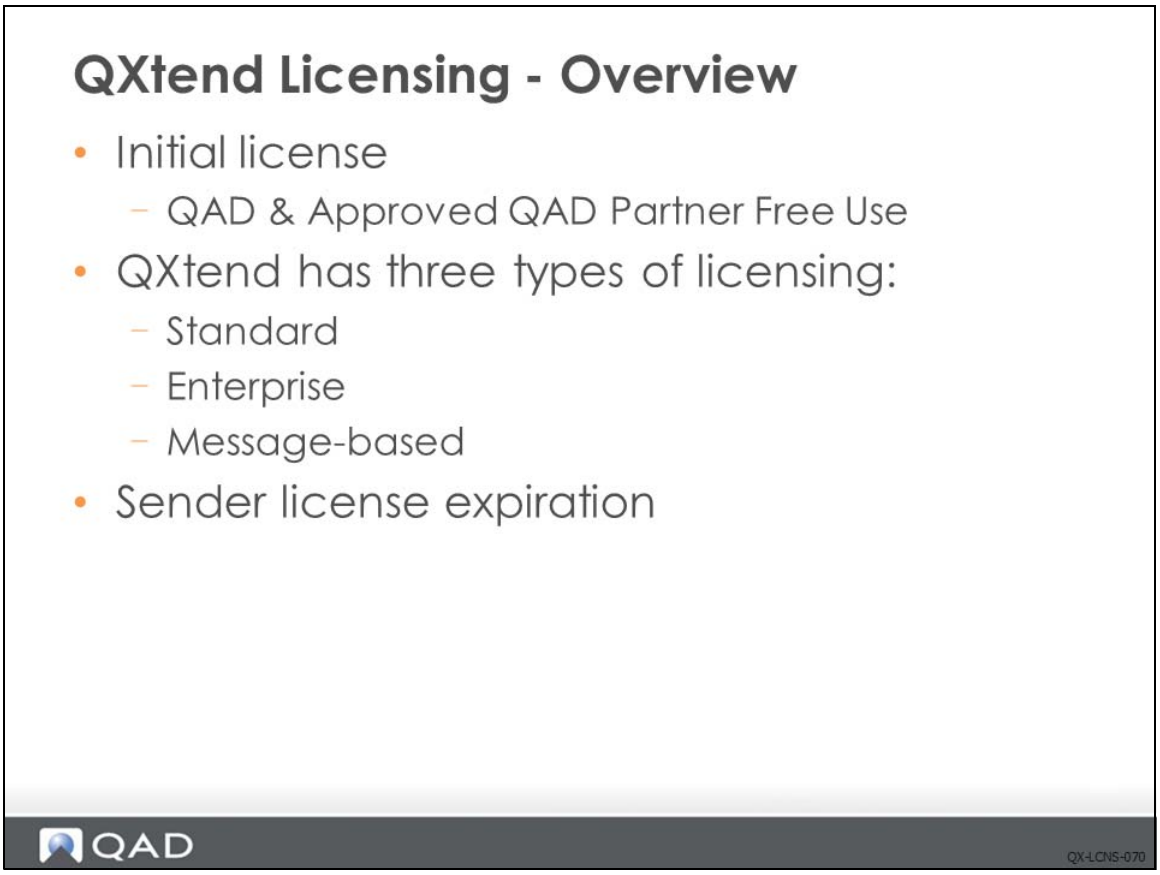
In the above example, there are four receivers.



Agents move data into and out of the system. Agents listen for requests to transmit data, gather the data to transmit, and deliver it to the receiver.

In QXtend Inbound, connection pools are assigned one or more processing threads. When the system processes a QDoc request, it uses an available connection pool agent.

In QXtend Outbound, Message sender sessions are the processing agents. Each agent is licensed, and a limit encoded into the license ensures that the number of agents is not exceeded across the QXtend installation. The number of receivers is not enforced.



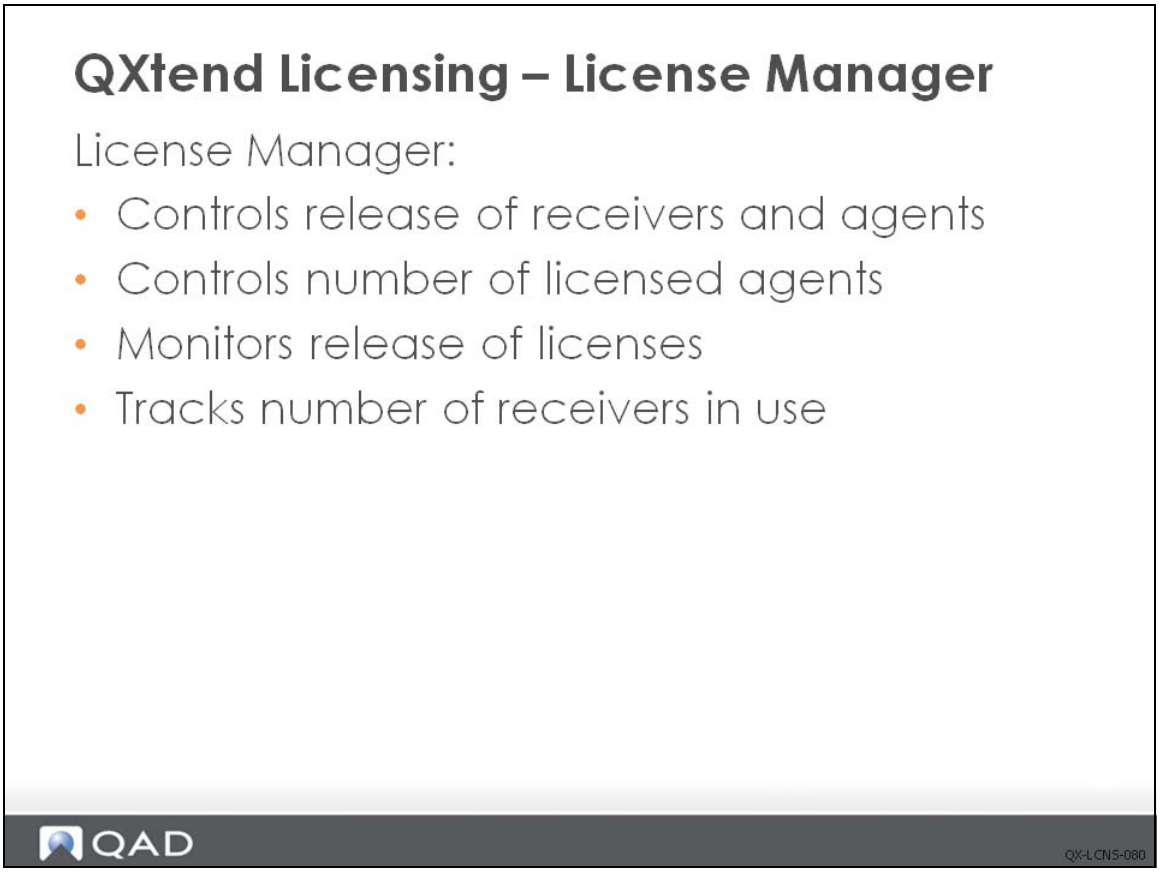
When QAD QXtend is first installed, the installation defaults to a QAD & Approved QAD Partner Free Use license. This license enables QAD QXtend to process data synchronization and QAD internal messages - including messages between QXO and QXI - free of charge.

QAD QXtend uses three types of licensing:

- Standard: Standard licensing limits the number of agents and receivers that can be used across QXtend.
- Enterprise: Enterprise licensing allows unlimited receivers and agents. It is generally used in implementations where the total number of receivers exceeds 47 or the total number of agents exceeds 68.
- Message-based: This licensing type is based on the number of messages processed, where the customer has bought a certain amount of QXtend messages.

Sender license expiration is used to control the use of QXtend products that rely on QXtend, for example, Excelerator. A sender license can be issued with an expiration date. Then, when the sender license expired, QXtend will no longer process the Qdoc from this sender. An error message will be returned indicating the license has expired.

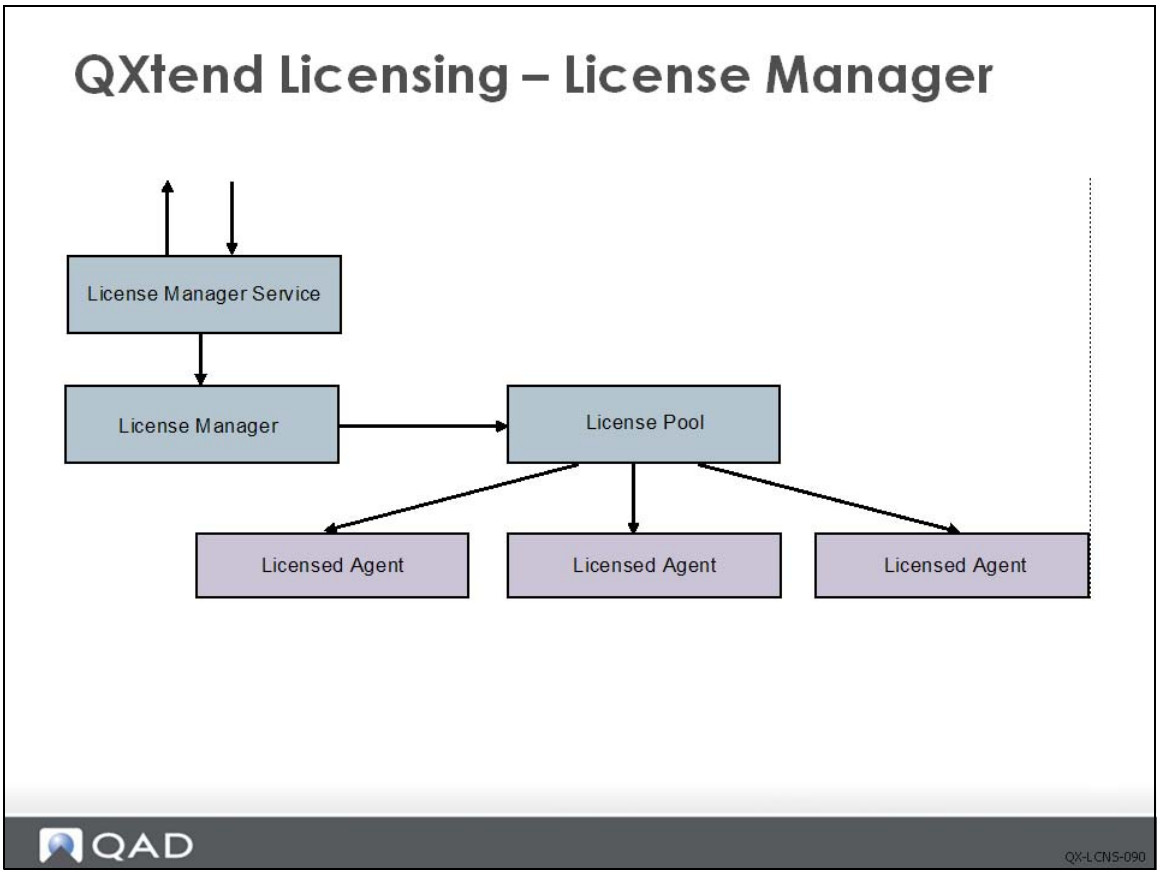
License Manager



The License Manager controls the release of receivers and agents; it also lets you limit the number of licensed agents, monitor the release of licenses, and track the number of receivers in use.

The License Manager enforces the agent limit encoded in the QXtend license by tracking when agents are requested and when they are released. It retrieves the agent limit by decoding the agent count from the license string. This enforced limit ensures that the number of agents is not exceeded across a QXtend installation.

When the License Manager is started, it creates a license pool for the current QXtend instance. The license pool then creates a pool of agents, determined by the licensed agent count, and initializes them. Agents in the initialized state have been created, but are not yet assigned a processing task.

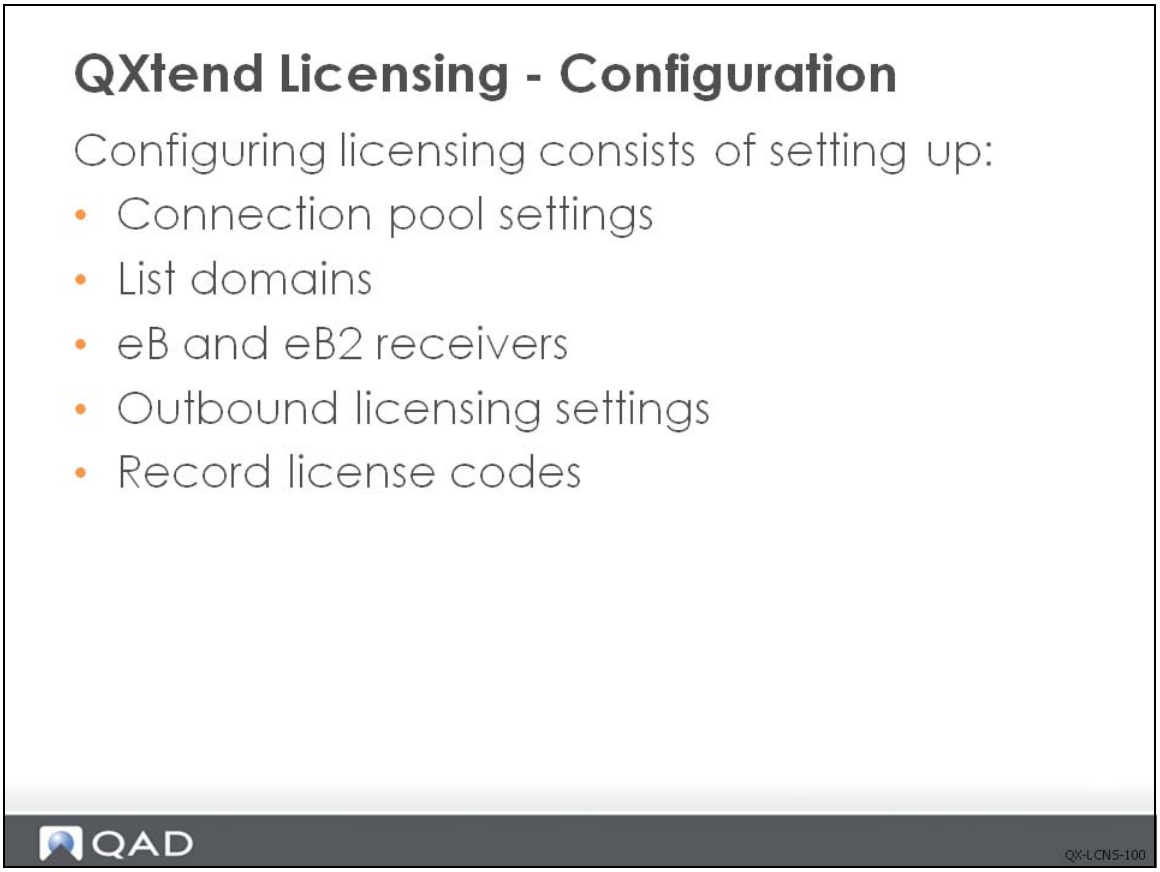


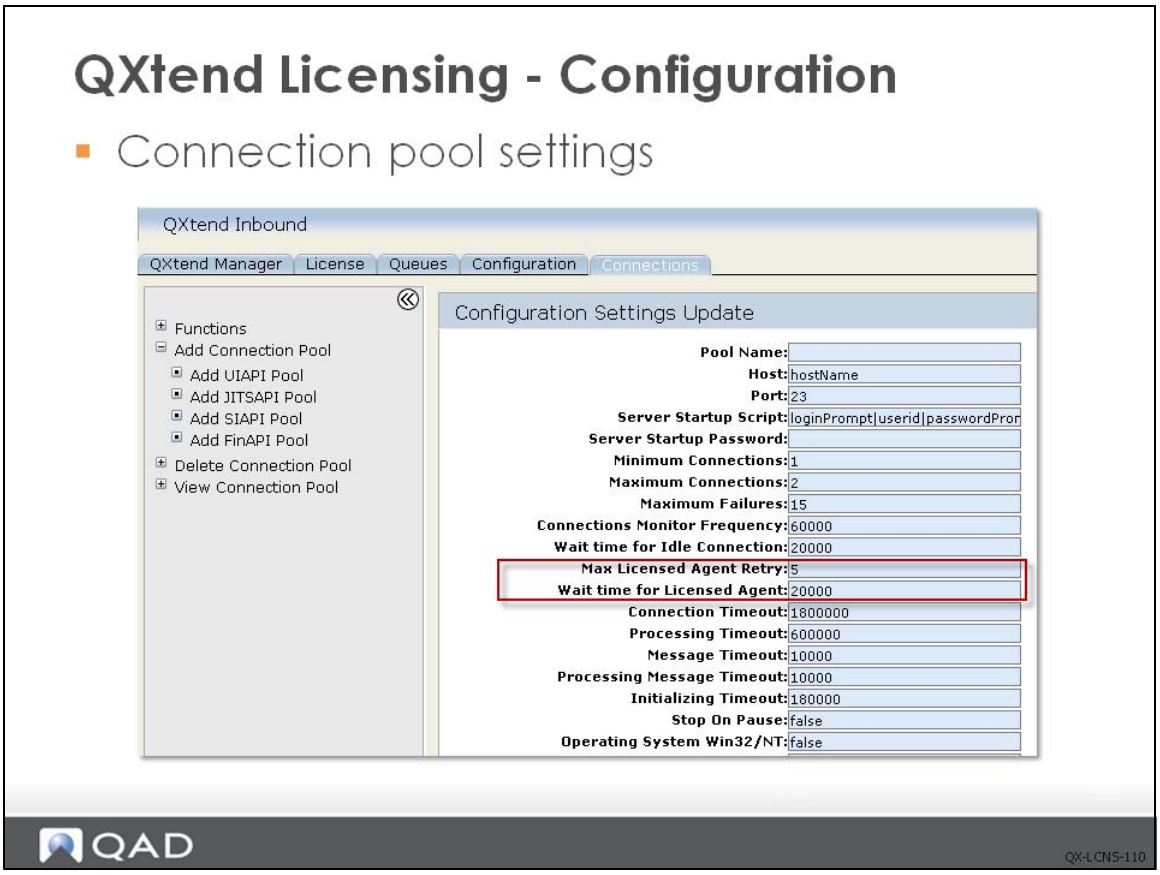
Each time a message sender service in QXtend Outbound is started for an unregistered subscriber, the system makes a call to the License Manager to reserve an agent for the task. In QXtend Inbound, connection pool agents process QDoc requests, and one or more processing threads are assigned to each connection pool.

Each time a QDoc is processed, a request for a licensed agent is sent to the License Manager. If there are no available agents, an exception is returned and processing stops. The system waits a predefined period for an available licensed agent. The waiting period is determined using a combination of the values in the Configurable Timeout and Maximum Retry fields in the Connection Pool window.

A licensed agent changes state from idle to busy when processing a task. When the task is complete, the agent becomes idle again and is available to process other tasks. This allows agents to be used freely across the QXtend Inbound and QXtend Outbound services.

Licensing Configuration





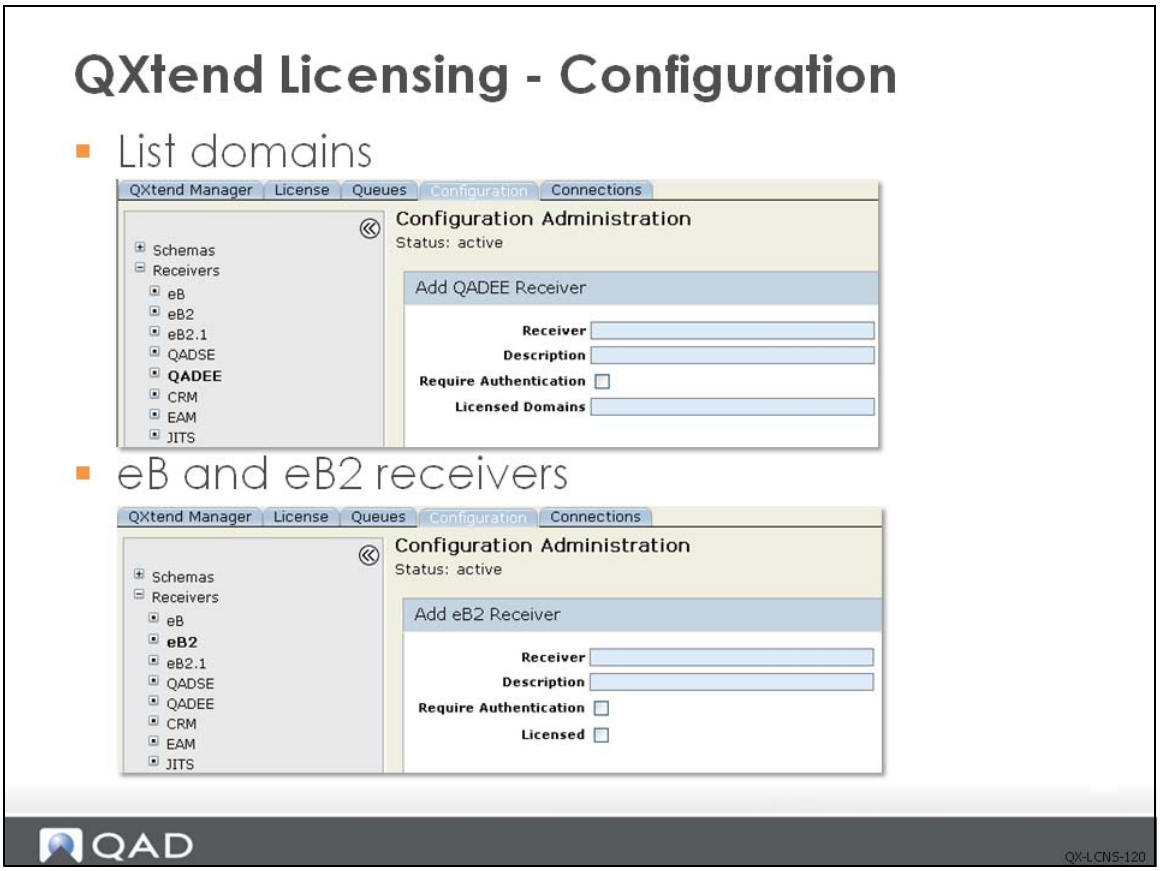
Connection Pool Settings

The Connection Pool window enables you to specify the number of attempts the License Manager makes to reserve a licensed agent, and the period of time the License Manager waits for an available agent. If the License Manager cannot reserve an agent and has exceeded the number of retry attempts, QXI returns an exception.

You need to configure the following settings:

Max Licensed Agent Retry. Specify the number of times the system attempts to reserve a licensed agent before returning an exception.

Wait Time for Licensed Agent. Specify the number of milliseconds that the system waits for a licensed agent.



List Domains

Use the Add Receiver window to assign a list of valid domain codes when you define a new receiver. The License Manager rejects any QDoc for an undefined domain.

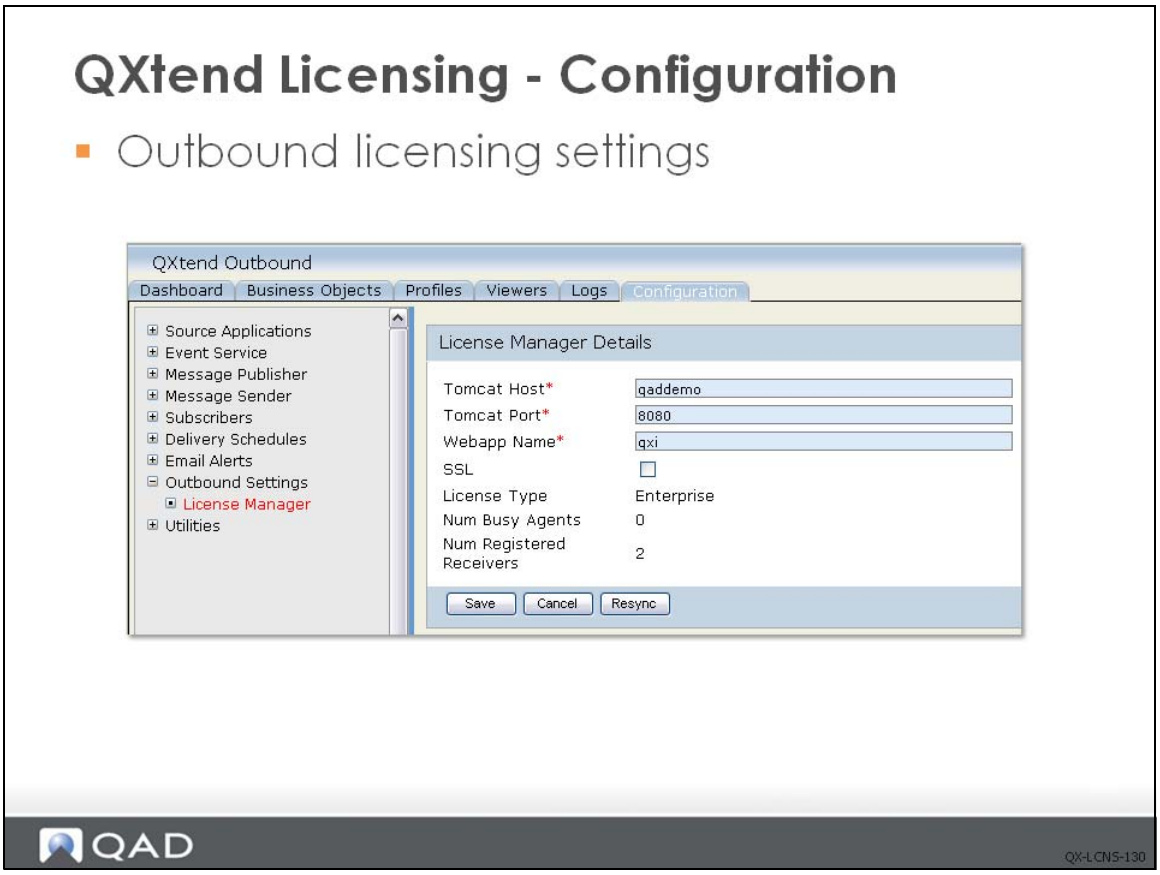
You can specify a comma-separated list of domains or a single domain. If you define a comma-separated list, the system converts the list into the appropriate XML format, and updates the `qdocReceivers.xml` file. If a receiver does not use domains, the `qdocReceivers.xml` file has a single domain entry that defaults to the receiver name.

Note If the domain is not specified in the request QDoc, then the default domain defined in the Connection Pool window is used. If a default domain is not specified on the connection pool, the request fails to process.

When you update the list of valid domains in the Add Receiver window, the receiver count is also updated in the License Manager.

eB and eB2 Receivers

When adding receivers for eB and eB2, the Add Receivers window does not show a list of licensed domains. You must select the Licensed field to indicate whether the receiver will be processing requests that require a licensed agent.



Outbound Licensing Settings

Use the License Manager Details window in the Configuration tab to configure the License Manager for QXO. This enables you to record the location of QXI to ensure that the count of licensed agents and receivers is synchronized across QXtend. Each time you configure a subscriber (receiver) in QXO, the system sends a message to the License Manager. Configure the following fields:

Tomcat Host. Specify the host name of the License Manager.

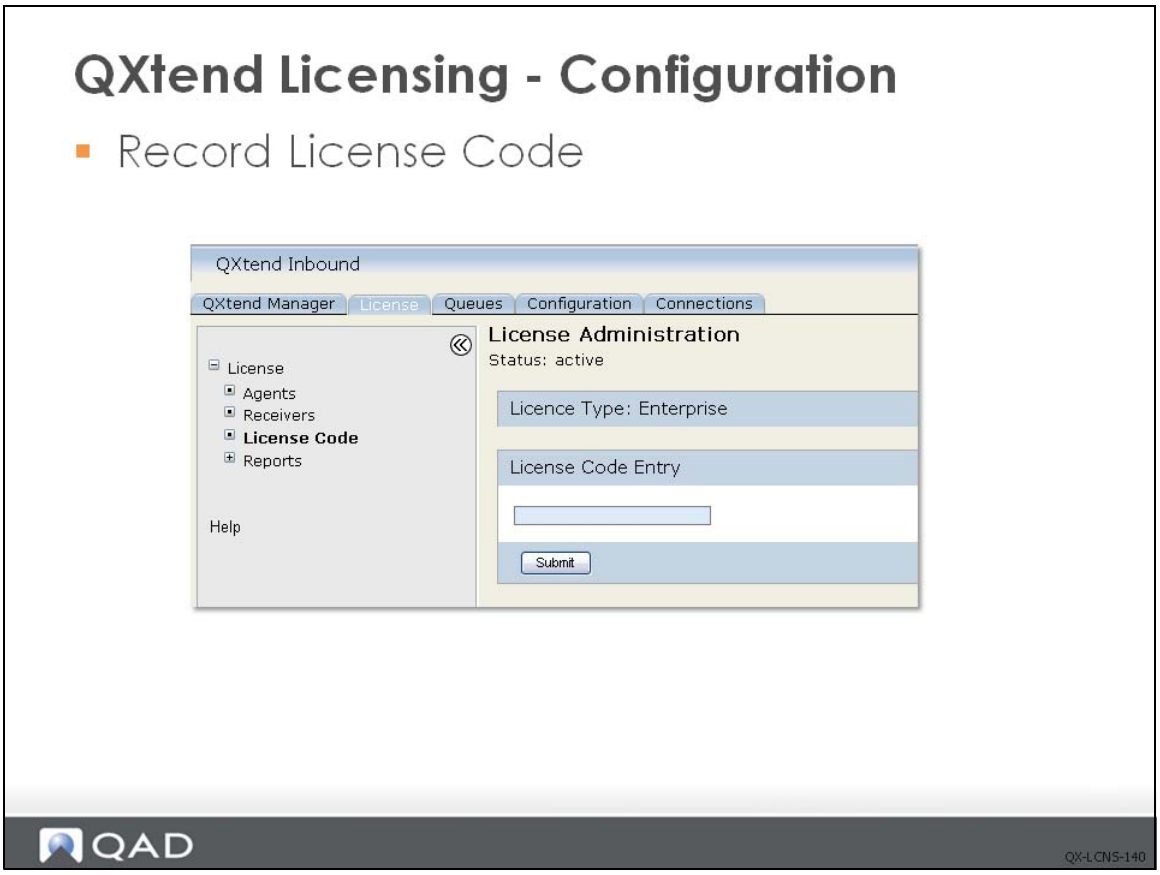
Tomcat Port. Specify the port where the License Manager resides.

Webapp Name. Specify the web application name of QXI.

License Type. Displays the type of license. The possible types are Standard, Data Synchronization, or Enterprise.

Number of Busy Agents. Displays the number of busy agents in both Inbound and Outbound.

Number of Registered Receivers. Displays the number of registered receivers in both Inbound and Outbound.



Record License Codes

When QAD QXtend is first installed, the installation defaults to a QAD & Approved QAD Partner Free Use license. This license only permits data synchronization and QAD internal messages to be processed. In order for QAD QXtend to communicate with other products, you must record a license code issued by QAD.

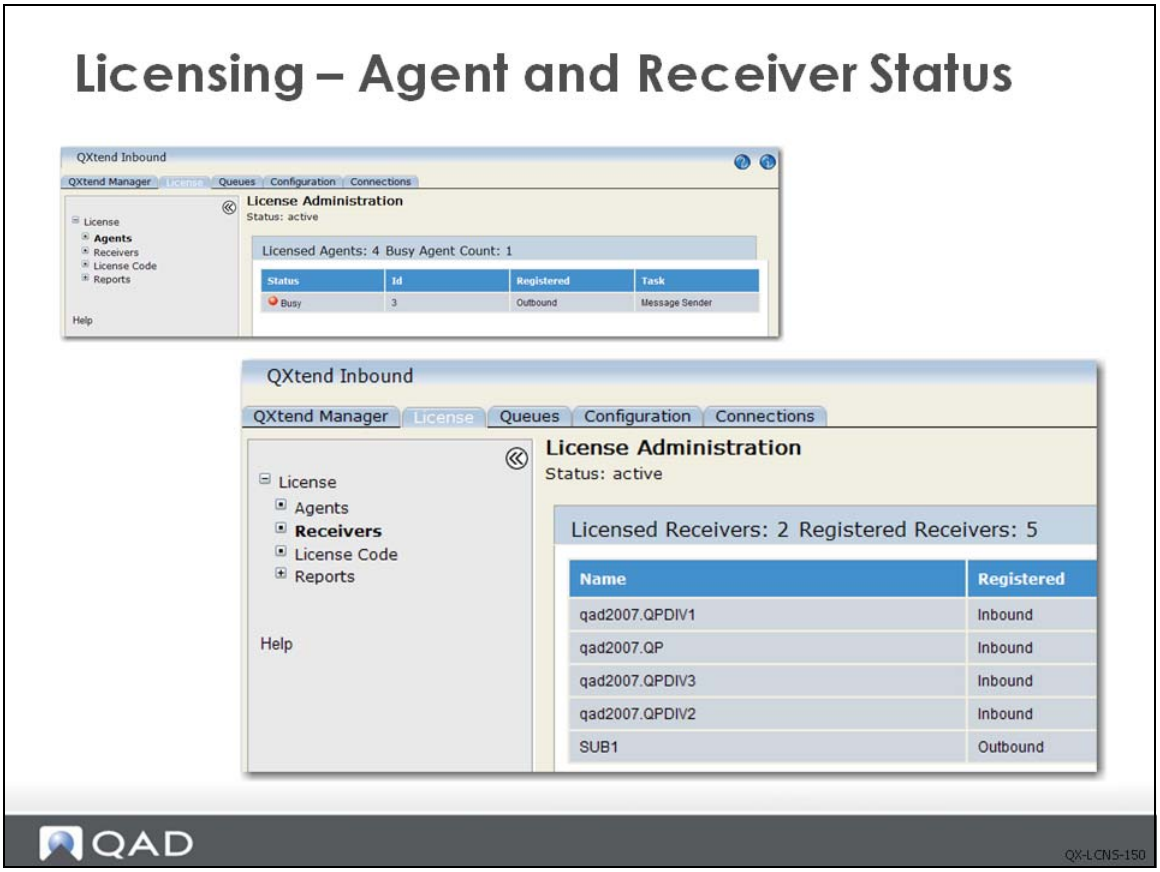
Use the License Administration window in QXI to submit a license code. Configure the following fields:

License Type. Displays the type of license.

License Code Entry. Enter the license code you received from QAD, and click Submit.

When you submit a new license code, the system validates it and stores the information in an encrypted XML file. The code has the number of agents and receivers encoded within it.If a license file has already been entered, the system overwrites it and resets the License Manager's internal count information.

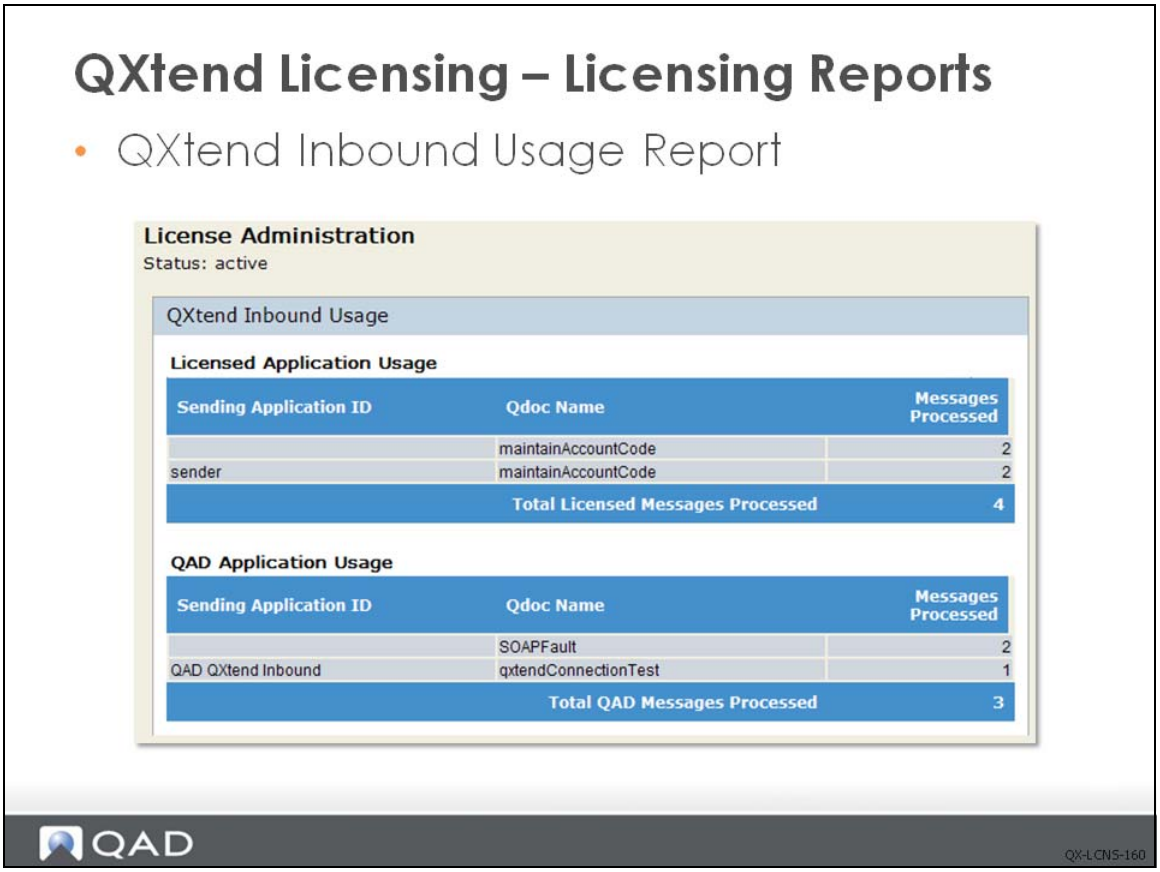
Agent and Receiver Status



The Licensed Agents window tracks and displays the number of licensed agents, as defined in the license code. It also displays the number of agents that are currently used in processing, and the task assigned.

Use the Licensed Receivers window to display the total number of licensed receivers in both QXtend Inbound and Outbound, and to view the list of registered receivers.

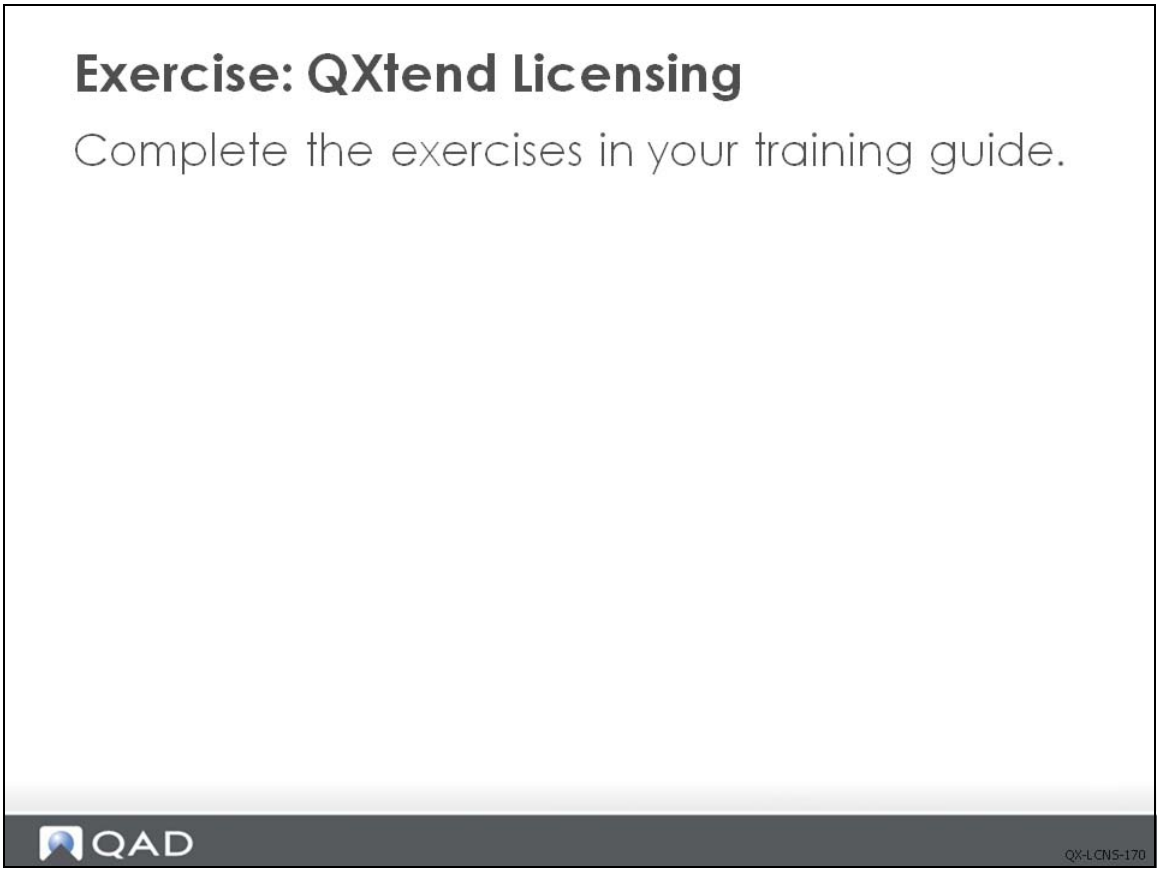
Licensing Reports



QXtend Inbound Usage Report

The QXtend Inbound Usage report displays the number of messages processed through QXI. The report indicates which messages originated from licensed applications and which are free QAD messages.

Exercise: QXtend Licensing



The following list shows a number of key concepts used in licensing in QXtend. In each statement below, fill in the correct term from the list.

license code	QXtend Inbound Usage Report
License Manager	QAD & Approved QAD Partner Free Use
message-based	receivers and agents
exception	free
not free	

1. In order for QXtend to communicate with other products, a _____ issued by QAD must be submitted.
2. A license code encodes the number of _____ that are available for each QXtend installation.
3. QXtend has three types of license: standard, enterprise, and _____.
4. When QXtend is first installed, it defaults to a _____ license.
5. The _____ controls the release of receivers and agents; it also lets you limit the number of licensed agents, monitor the release of licenses, and track the number of receivers in use.
6. If, during the processing of a QDoc, there is no agent available, the system returns an _____ and processing stops for that QDoc.

7 _____ allows you to monitor messages processed.

8 Transactions from QAD EAM to QADEE through QXtend are _____; data synchronization between QADEE and third party product through QXtend are _____.

Lab: QAD QXtend Licensing

QAD QXtend 1.7.2 has been installed with both QXI and QXO components. In the previous labs you performed the basic configuration that is required to enable requests to be processed by QXI and QXO. In the following lab exercises you will see the impact of the different QAD QXtend licenses on the processing within QAD QXtend.

`<LabHomeDirectory> = C:\QXtendTraining\Labs\04-Licensing\`

The QAD QXtend instance you have been using for the labs so far does not have a specific license installed. When no license is installed, you get the default license "`QAD & Approved QAD Partner Free Use`", which restricts the types of transactions that can be processed. The following lab will show these restrictions, describe how to install new licenses, and explain the configuration that is required depending on the license type.

1. License Exceptions

In the previous lab exercises you could process requests to QXI without receiving any exceptions. No exceptions were encountered because of the sender ID `qxtendtrain` that has been used on each of the requests. This sender ID is valid within QAD QXtend and allows any messages to be processed. If we process requests without a sender ID, license exceptions will be returned when processing the requests.

1.1 QXI Licensing Exception

Create a new soapUI Workspace and soapUI.

1 Open soapUI on the Windows image, use the shortcut on the Desktop.



2 Create a new workspace

   a File – New Workspace.

   b Set the workspace name to Lab 04 – Licensing.

   c Save the workspace file in <*LabHomeDirectory*>.

3 Create a new WSDL project.

   a Right-click the workspace name.

   b Select the New soapUI Project.

   c Set the project name to `Licensing 04`.

   d Set the WSDL file to:
   <*LabHomeDirectory*>`/Schemas/maintainCustomerItem-ERP3_1.wsdl`

   e Click OK to create the project.

4 Drill down to the Request 1 message created under the Licensing 04 project.

5 Right-click Request 1 and select the Clone Request option. Enter a name for the new request.

6 Edit the new request so that it can be processed by QXI. Change the SOAP header:

   a Set the receiver to `QADERP`.

   b Leave the Sender ID blank.

   c Set `suppressResponseDetail` to `true`.

7 Create the necessary session context entries:

   a Create two `ttContext` iterations.

   b Qualifier = `QAD`, Name = `version`, Value = `ERP3_1`.

   c Qualifier = `QAD`, Name = `domain`, Value = `10USA`.

Note If you copy and paste the empty `ttContext` node and blank lines are created, right-click the request and select the Format XML option. soapUI will remove any blank lines and correct the indentation of the XML.

8 Edit the application data section of the message:

   a The `dsCustomerItem` node by default will contain all the fields available when processing a Customer Item Master update. However, only a few fields are required for a transaction to process successfully. The best way to identify the mandatory fields is to run through the UI and see which fields you have to enter data for.

   b Edit the XML and create a new Customer item QDoc that will be processed successfully by QXI. Use the following values:

cpCust	10C1000
cpCustPart	SC-2134-Z
cpPart	01010
cpComment	QXtend Training Customer Item
cpCustPartd	Standard Connector
cpCustEco	35GY-001

9 Process the QDoc message that you have created in the new soapUI project.

10 Review the response. You will see that a licensing exception was returned by QXI. The exception should be `QXtend is not licensed for this type of QDoc Message`.

2. Standard License Type

The standard QAD QXtend license restricts the number of receivers and agents available to QAD QXtend to process transaction from external systems.

2.1 Load Standard License Code

The license codes that are delivered for QAD QXtend have data encoded in them that controls the number of agents and receivers that are available for that customer.

Load a QAD QXtend license code:

1. Open the QXI Web application in Internet Explorer:

   `http://qaddemo:8080/qxi`

2. Select the License tab.
3. Select the License node on the menu tree.
4. Select the License Code option. You will see `License Type: QAD & Approved QAD Partner Free Use`.
5. In the License Code Entry field enter the license code from the `License.txt` file on the Windows Desktop.
6. Click the Submit button.
7. Select the License tab.
8. Select the License node on the menu tree.
9. Select the License Code option. You will see `License Type: Standard`.

Reprocess the transaction created in soapUI in Section 1.1

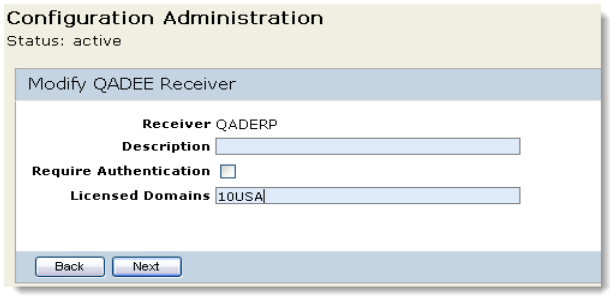
1. Reprocess the request. A different exception is received:

   `No Domain in this Receiver is licensed.`

2. Each Receiver must maintain a list of domains that can process licensed transactions.
   a. Open the QXI Web application in Internet Explorer:

      `http://qaddemo:8080/qxi`

   b. Select the Configuration tab.
   c. Select the Receivers node on the menu tree.
   d. Select the QADEE option.
   e. Select the check box for the QADERP receiver.
   f. Click the Modify button.
   g. Select the Continue Configuration update without suspending QXtend Inbound option and click the Submit button.
   h. Enter `10USA` into the Licensed Domains field and click the Next button.



Reprocess the transaction created in soapUI in section 1.1

1 Reprocess the request. This time the request processes successfully.

2 Validate that the transaction has been processed and that the customer item has been created.

3. QXO Licensing

Certain messages processed by QAD QXtend are processed free of charge and do not require a license code:

- When the subscriber type is QAD QXtend. In this case, sending messages from QXO to QXI is considered a free operation and thus not restricted by the number of licensed agents.
- When other (valid) applications are registered in QXI. In this case, messages can be sent to a Web service or file drop.

Every file drop/standard Web service subscriber that is created is included in the receiver count, and each message sender agent consumes a licensed agent.

Add a File Drop Subscriber to QXO

1 Open the QXO Web application in Internet Explorer:

`http://qaddemo:8080/qxo`

2 Select the Configuration tab.

3 Select the Subscribers node on the menu tree.

4 Click the New button.

5 Use the following values for the subscriber configuration:

Subscriber Code	FileDrop
Source Domain	10USA
Communication Method	File Directory Service
XML Syntax	QDoc 1.1
Select the Allow Superseded check box	
Target Directory	/dr01/temp/qdocs

a. Do not select the Include Soap Envelope check box.

b. To register the profile messages that are sent to the subscriber, click the Register Profiles button.

c. Ensure the App Type is set to QADEE.

d. Select the fetch button next to the Filter prompt to get a list of the available profiles.

e. Select the QADEE/GeneralizedCode/GeneralizedCode profile.

f. Click the Add button.

g. Click OK to return to the subscriber configuration screen.

h. Click the Save button.

Create a new message sender and add the FileDrop subscriber to the Message Sender MS2:

1. Open the QXO Web application in Internet Explorer:
   `http://qaddemo:8080/qxo`
2. Select the Configuration tab.
3. Select the Message Sender node on the menu tree.
4. Click the New button.
5. Call the Message Sender `MS2`.
6. Click the Lookup button to register subscribers with the sender.
7. Select the FileDrop subscriber from the list.
8. Click the Add button.
9. Click OK to return to the sender configuration screen.
10. Click the Save button.

To see the effect of the changes, the QXO services need to be restarted

1. Open the QXO Web application in Internet Explorer:
   `http://qaddemo:8080/qxo`
2. Select the Dashboard tab.
3. Select the Services node on the menu tree.

4 Click the Stop All button to stop all QXO services.

5 Click the Start All button to start all QXO services.

4. Monitoring License Usage

The License pages of QXI allow you to monitor the use of agents and receivers:

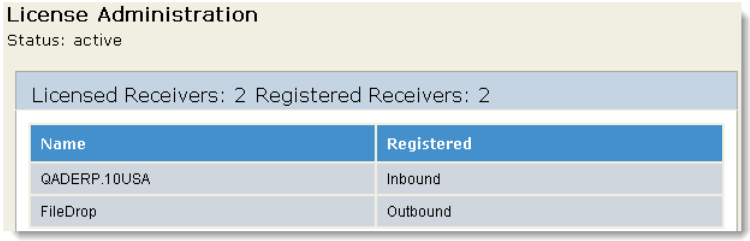
1 Open the QXI Web application in Internet Explorer:

```
http://qaddemo:8080/qxi
```

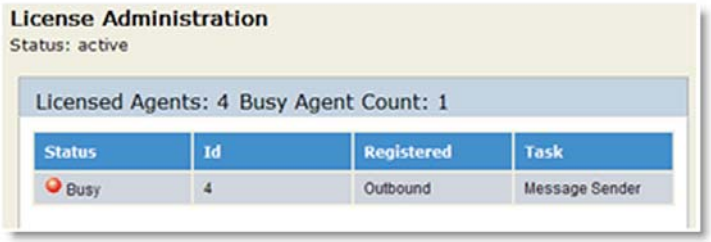
2 Select the License tab.

3 Select the License node on the menu tree.

4 Select the Receivers option. The number of receivers currently used is displayed and where they are registered (QXI or QXO).



5 Select the Agents option to see how many agents are currently busy.



QAD QXtend Inbound and Outbound both track the number of messages that have been processed. In QXI you can view the number of messages that have been processed by:

1 Open the QXI Web application in Internet Explorer:

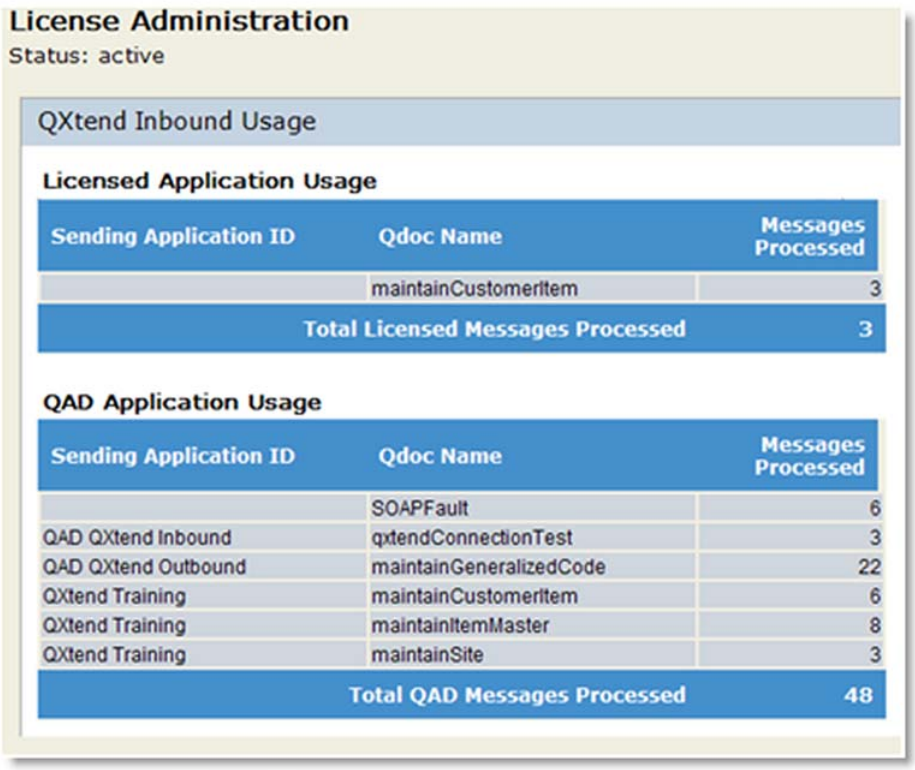
```
http://qaddemo:8080/qxi
```

2 Select the License tab.

3 Select the Reports tab.

4 Select the QXtend Inbound Usage option.

5 Click the Get Usage button. The transaction count for messages processed displays.



License Administration

Status: active

QXtend Inbound Usage

Licensed Application Usage

Sending Application ID	Qdoc Name	Messages Processed
	maintainCustomerItem	3
Total Licensed Messages Processed		3

QAD Application Usage

Sending Application ID	Qdoc Name	Messages Processed
	SOAPFault	6
QAD QXtend Inbound	qxtendConnectionTest	3
QAD QXtend Outbound	maintainGeneralizedCode	22
QXtend Training	maintainCustomerItem	6
QXtend Training	maintainItemMaster	8
QXtend Training	maintainSite	3
Total QAD Messages Processed		48

5. QAD QXtend Enterprise License

The Enterprise license type allows you to process requests without restrictions through QXI and QXO. With the Enterprise license there is no need to configure the list of licensed domains because the number of receivers being used and the number of agents does not have to be tracked.

5.1 Process Test Request

With the standard license, process a request to the 11CAN domain without adding the domain to the list of licensed domains.

1. Right-click the request created in Section 3 and select the Clone Request option. Enter a name for the new request.
2. Edit the new request so that it can be processed by QXI. Change the SOAP header:
   a. Set the receiver to be `QADERP`.
   b. Leave the Sender ID blank.
   c. Set `suppressResponseDetail` to `true`.
3. Create the necessary session context entries:
   a. Create two `ttContext` iterations.
   b. Qualifier = `QAD`, Name = `version`, Value = `ERP3_1`.

c Qualifier = `QAD`, Name = `domain`, Value = `11CAN`.

Note If you copy and paste the empty `ttContext` node and blank lines are created, right-click the request and select the Format XML option. soapUI will remove any blank lines and correct the indentation of the XML.

4 Edit the application data section of the message:

a The `dsCustomerItem` node by default will contain all the fields available when processing a Customer Item Master update. However, only a few fields are required for a transaction to process successfully. The best way to identify the mandatory fields is to run through the UI and see which fields you have to enter data for.

b Edit the XML and create a new Customer item QDoc that will be processed successfully by QXI, use the following values:

cpCust	10C1000
cpCustPart	2134-Y
cpPart	01010
cpComment	QXtend Training Customer Item
cpCustPartd	Standard Connector
cpCustEco	35GY-001

5 Process the request from soapUI. The response returned has this licensing error:

```
Request's domain is not in receiver's licensed domain list
```

5.2 Load Enterprise License Code

The license codes delivered for QAD QXtend have data encoded in them that controls the number of agents and receivers that are available for that customer.

To load a QAD QXtend license code:

1 Open the QXI Web application in Internet Explorer:

```
http://qaddemo:8080/qxi
```

2 Select the License tab.

3 Select the License node on the menu tree.

4 Select the License Code option. The indicator `License Type: Standard` is displayed.

5 In the License Code Entry field enter the license code from the `License.txt` file on the Windows Desktop, and click the Submit button.

6 Select the License tab.

7 Select the License node on the menu tree.

8 Select the License Code option. The indicator `License Type: Enterprise` is displayed.

5.3 Process Test Request

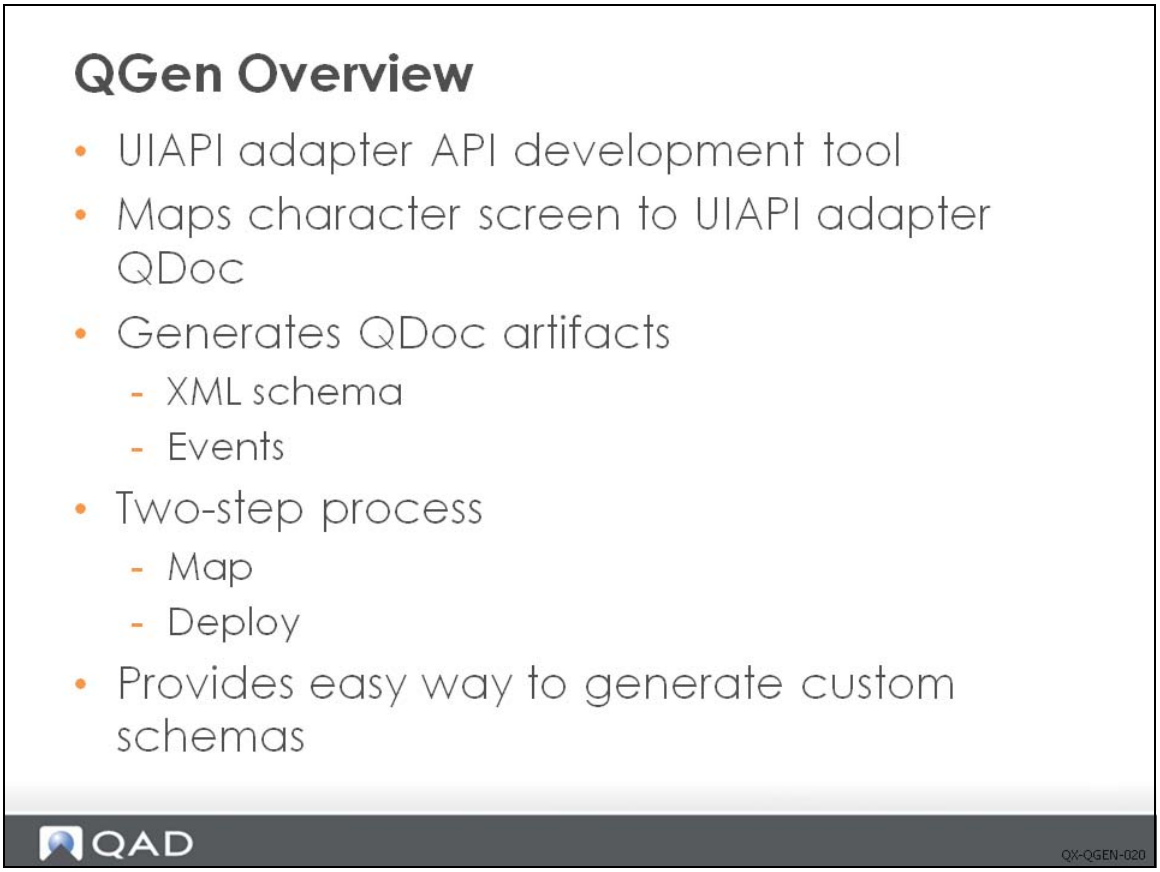
Now that you have loaded an Enterprise type license, there are no restrictions on the types of messages that can be processed. You can process a request to any domain without receiving any licensing exceptions. The QADERP receiver currently only has the 10USA domain listed as a licensed domain. You will now reprocess the Customer Item create message from Step 5.1.

1. Process the create Customer Item message used in step 5.1 using soapUI. QXI successfully processes the message and returns a `Success` message.
2. Verify that the new Customer item has been created in the 11CAN domain.

Chapter 5

QGen

QGen Overview

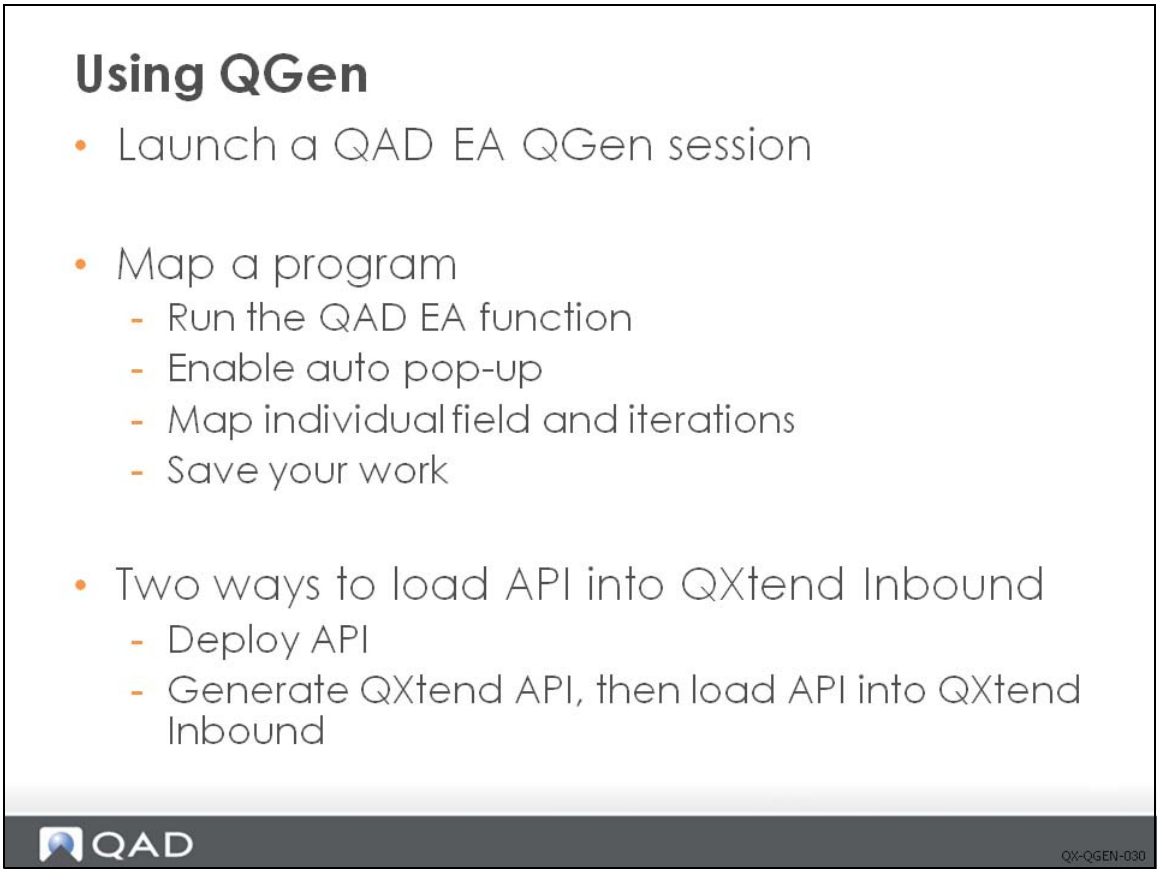


QGen is a development tool that facilitates the creation and maintenance of QXtend APIs that leverage the UI API adapter within QXI. The UI API adapter uses the character screen within the application to load data. Therefore you can only use QGen on functions that have a character version of the interface. QGen records the structure and field information from QAD EA character UI screens and saves the information to data files. Not all of the information required for each field is available from Progress, so you have to supply some specific information.

The data recorded by the mapping process is then passed into the QGen generation engine, which transforms the data in the files into the relevant QDoc XML schema and events files, and deploy them to the custom area of QXI to make them available for processing.

QGen makes it simple for services to create a new API. QGen can handle most customizations a customer makes to their system. However, the UI cannot have any widgets that are not supported by the XML engine. If a QDoc exists for a function but the customer has modified a function by adding new fields and/or frames, the data file can be extracted from the QXtend server and loaded in the QGen datFiles directory. Then the new information can be recorded from the customized screens and a new version of the schema recorded. The new schema (and potentially, events files) must be uploaded to the QXtend server, and then the customized function is accessible from the QXtend server.

Using QGen



Using QGen is easy, though you must follow several rules to ensure that you generate the correct API definitions. The following slides describe how to create a new UI API adapter API.

QGen must run in a character OpenEdge session—you cannot run it from the QAD .Net UI or from an OpenEdge GUI session. A QGen-specific startup script is required to launch a QGen session. The startup script runs the `<qxi-adapter>/tool/runProgramMapper.p` program before calling the standard QAD startup routine `mf.p`. When a QGen session is launched, it looks and functions just like a QAD EA character session.

Once in QGen, do the following to map a program:

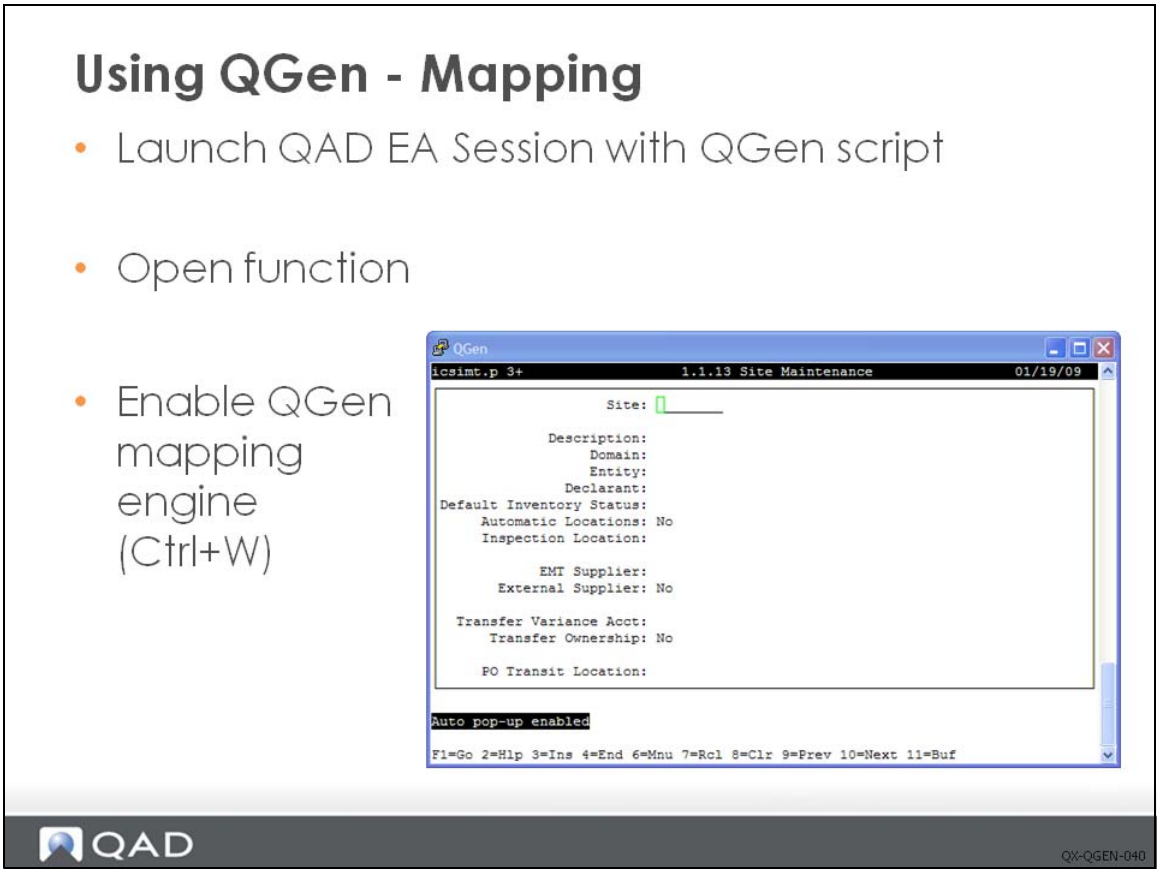
1. Run the function. The function you are mapping is launched from the QAD EA menu.
2. Enable the auto pop-up. The auto pop-up feature of QGen is enabled. This initializes QGen and displays a pop–up frame after each field in the UI is accessed to allow you to record specific field information required to generate the new UI API adapter artifacts.
3. Map the screen. Enter data into the application screen. As you enter the data, record the field information required by the QGen pop-up screen. QGen records the fields as you move through the screen, as well as the screen navigation to move from frame to frame. Ensure that you map all fields in the function before completing the mapping; this might need multiple passes through the application screen.

4 Save. When all information on the application screen has been recorded, save the information. This creates a file in the QXI adapter directory that contains all of the information recorded during the screen mapping.

From within QGen you can now deploy API to QXtend Inbound using the Generate Docs function. The Generate option in QGen lets you generate QDocs that use either the QDoc 1.0 or QDoc 1.1 standard. The QDoc 1.0 option is provided for backward compatibility only—*you should use the QDoc 1.1 standard for all new QDocs*. The artifacts generated by QGen are stored in the QXI adapter installation directory.

The QGen also provides you an option to output all the artifacts required by QXtend Inbound, then load the new API into QXI. To do this, copy the generated files from the server to the client machine, uploading them to QXI. In the QXI Configuration Manager, use the Schemas node on the tree menu to upload the files generated from QGen into QXtend as a UI API adapter interface.

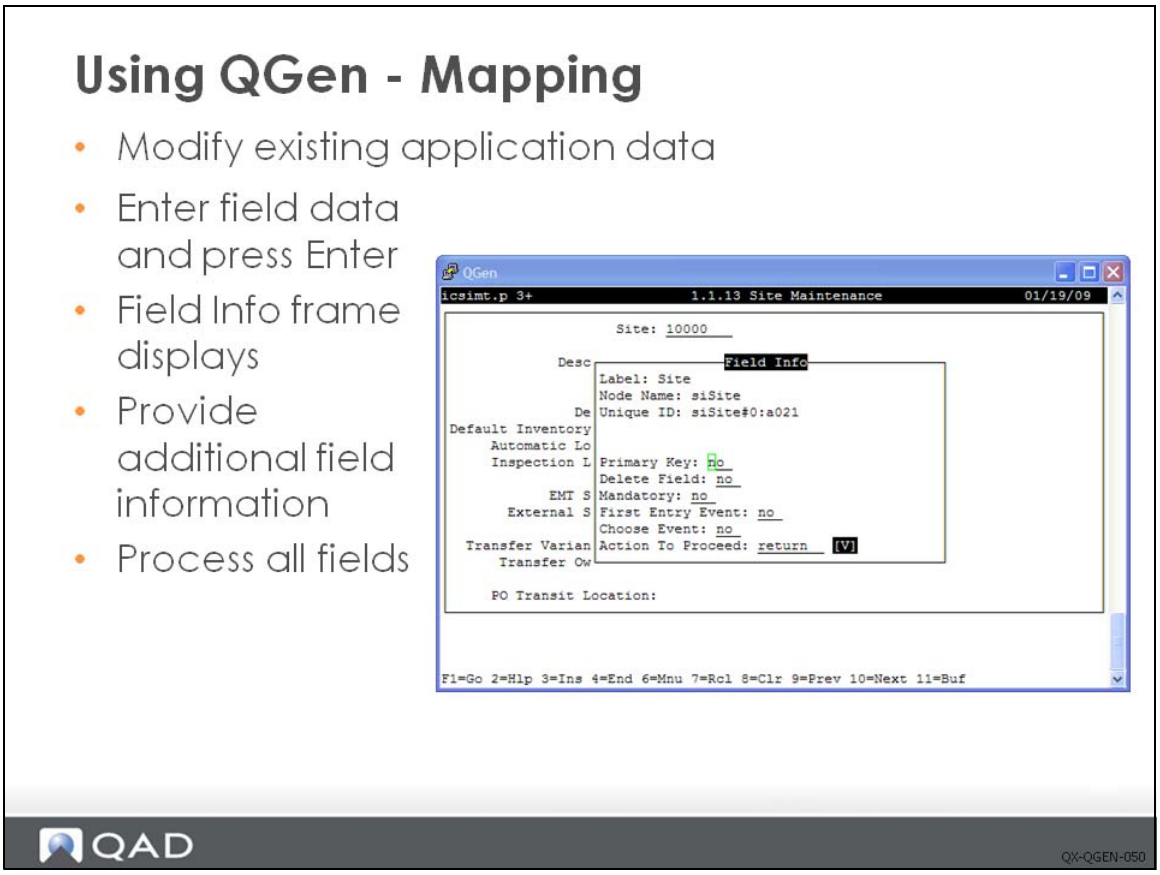
Mapping



Using the QGen startup script, launch a QAD EA character session. The QGen startup script executes the `tool/runProgramMapper.p` program before launching `mf.p`. The `runProgramMapper.p` program initializes QGen and sets up session triggers that drive the QGen field information pop-up screens and the QGen menu.

From the QAD EA menu, launch the function for which you want to create a UI API adapter API. This displays the normal character window that you would expect to see when using a character screen to enter data.

Before entering any application data, initialize QXtend by pressing Ctrl+W. The message "`Auto pop-up enabled`" displays in the bottom-left corner of the window. QGen is now enabled and will record any actions you perform in the application.



The easiest way to map a function the first time is to walk through an existing entry that has already been entered. This way you will know which specific information you will require to complete the mapping. You must walk through each field in the function. Press Enter to leave each field, not F1.

Note If pressing Enter does not bring the Field Info pop-up, you might have to use Ctrl+F to bring that pop-up.

Only information from the fields you access is recorded and added to the API. After entering data into a field and pressing Enter, the QGen Field Info pop-up frame displays. This frame shows information about the field and prompts you for further details that are used to control the processing when called from QXtend. The Field Info frame prompts for the following values:

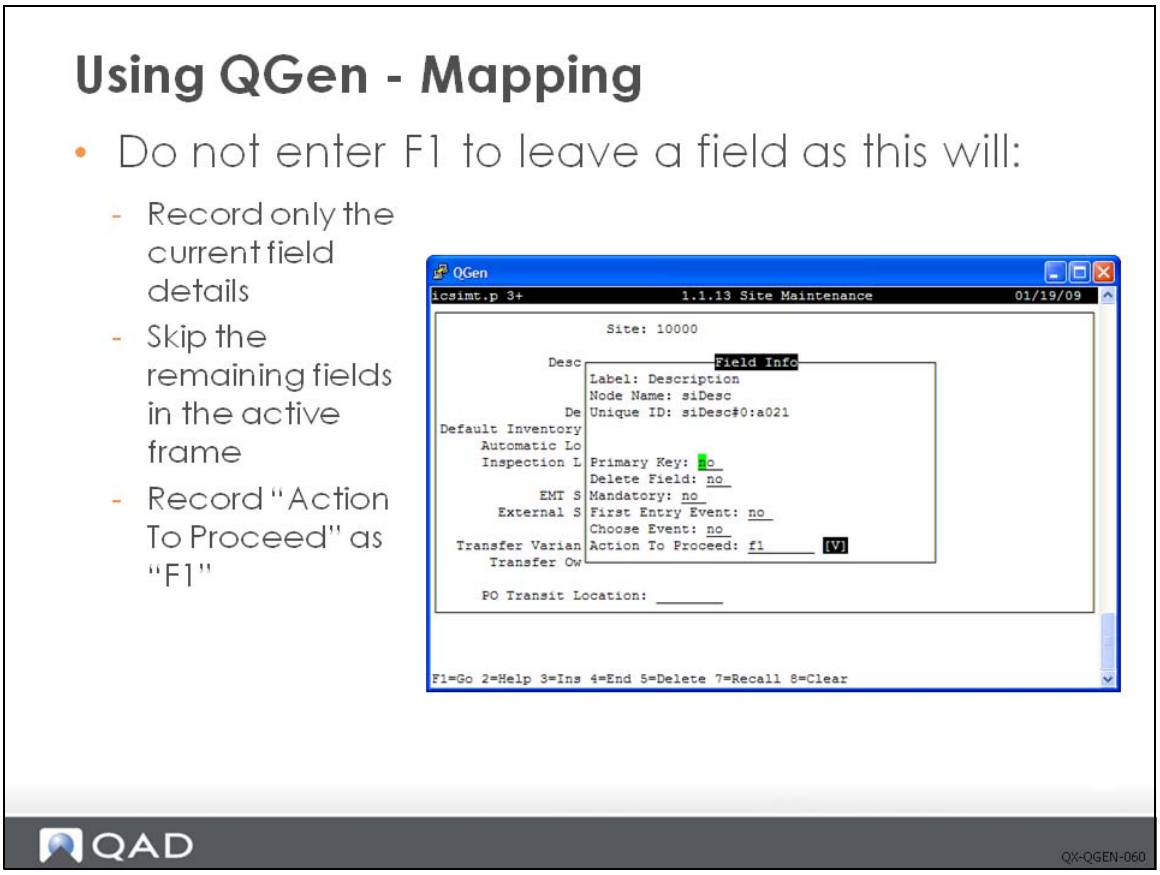
Primary Key. Logical field Enter Yes if this field is part of the primary index in the database, or if you want to include this field in the response message that is returned from QXI.

Delete Field. Logical field Enter Yes if the Delete key F5 is available on the field.

First Entry Event. Logical field Enter Yes if a special action should be performed the first time this field is encountered when processing a QDoc. The best example for this event is Sales Order Entry and the Line field. If we want to ensure that multi-line mode is always used when processing sales orders, we would send F4 – m – F1. If Yes is entered, an extra frame displays when F1 is pressed in this frame.

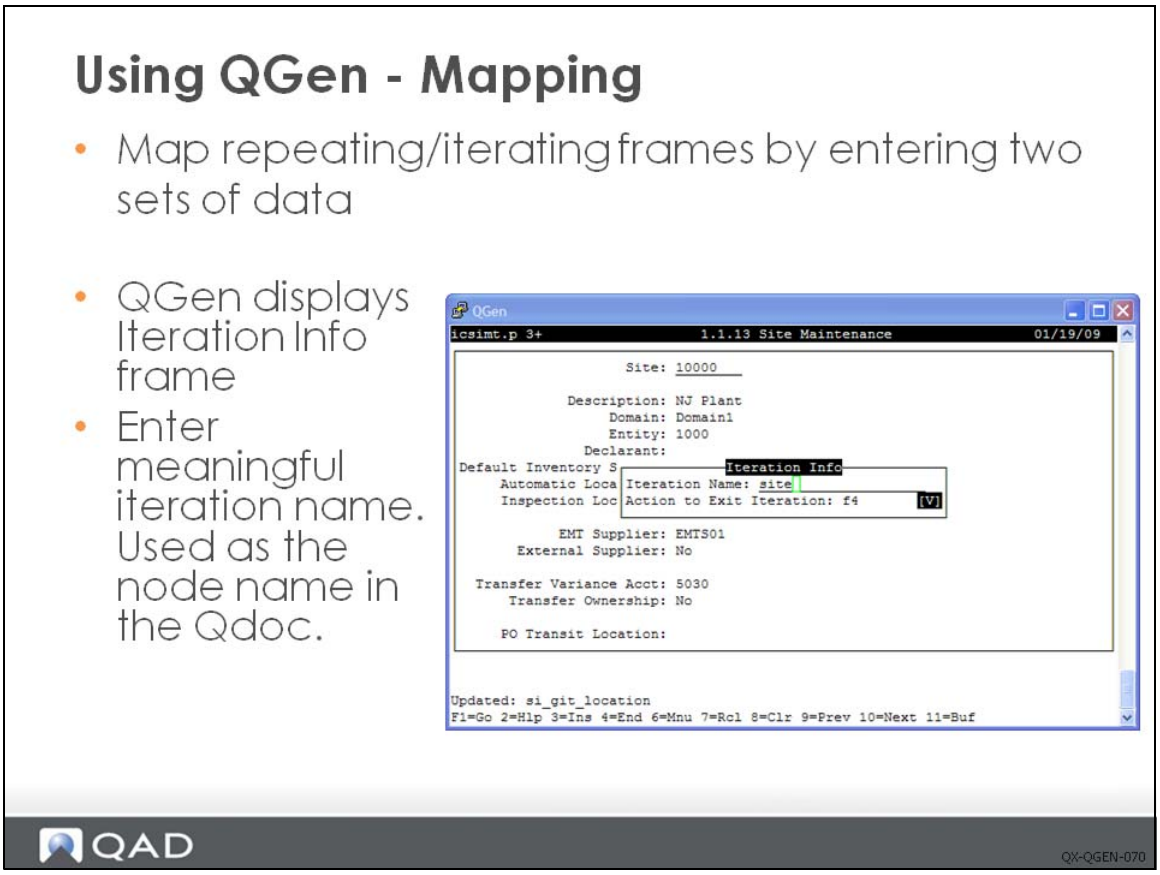
Action To Proceed. The key to use to navigate from this field to the next field/frame. The user must choose from a drop-down list. The default value is Enter. However, sometimes Enter is not used and special navigation is required. This field lets you define the action required to leave this field. For example, when recording transaction comments you must press F4 to continue entering application data.

When all data has been entered, press F1. You are taken either to the First Entry Event frame or to the next field.



You must walk through each of the fields in the function. Remember, leave each field by pressing Enter, not F1.

Note When generating a custom API you might sometimes want to skip an entire frame. In this case you would press F1. However, you will not be able to load any data into the fields you have skipped.

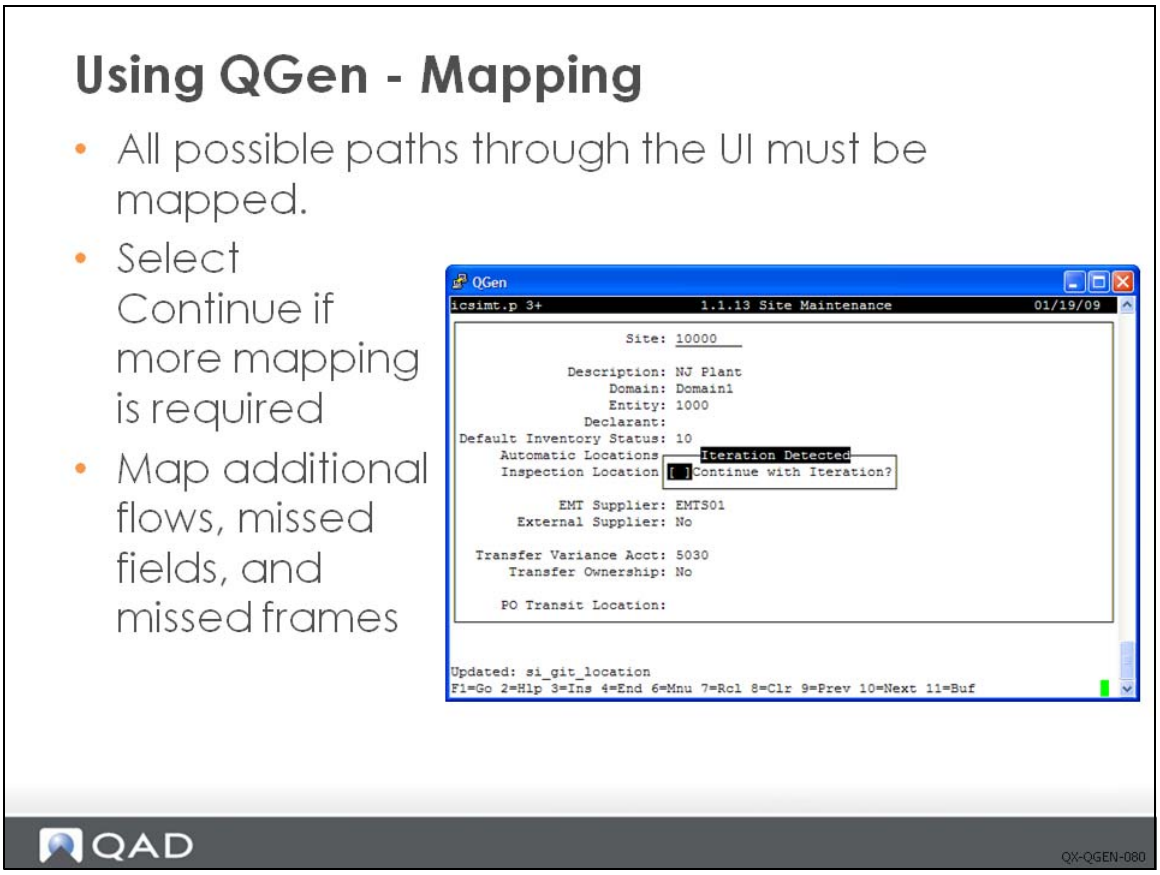


Many screens in QAD EA have repeating sections of data. These iterating sections typically represent parent-child relationships in the application, and these data iterations must be mapped correctly when generating an API for QXI. Each iteration that is mapped represents a different level of data in the generated QDoc; each iteration can occur many times in a QDoc request instance.

Ensure that on frames where it is possible to iterate through the screens that you go through at least twice; doing so ensures that you pick up the iteration name.

The second time you go through the screen, QGen recognizes that the field has been encountered before and identifies it as an iteration. When you encounter an iteration, you must specify the name to use in the QDoc to identify the iteration—for example, salesOrder, salesOrderDetail. The names given to the iteration should describe the iteration data clearly as this will affect how understandable the XML documents are.

The Action To Exit Iteration provides the navigation keys required to exit the current iteration and enter the next screen, or to exit the function. Choose a value from the drop-down list.

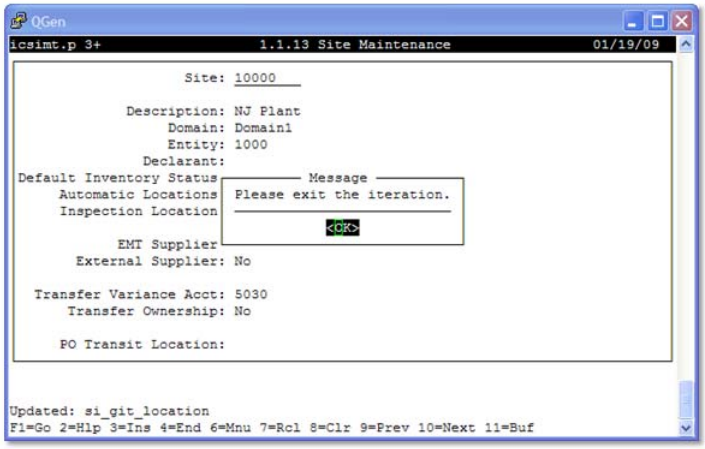


Typically, you cannot map complex functions in QAD EA in a single pass, as various control settings and data values cause extra screens and fields to display. To correctly map the entire function, you must run through the transaction several times to ensure that you map all necessary frames and fields.

Once the iteration name has been entered, the system asks if you want to continue with the current iteration. If some fields have not been mapped—or if there is another route through the screen that has not been mapped—select the toggle button and repeat the iteration, this time choosing a different flow. The second time through the function, the system only prompts for fields and iterations not already mapped.

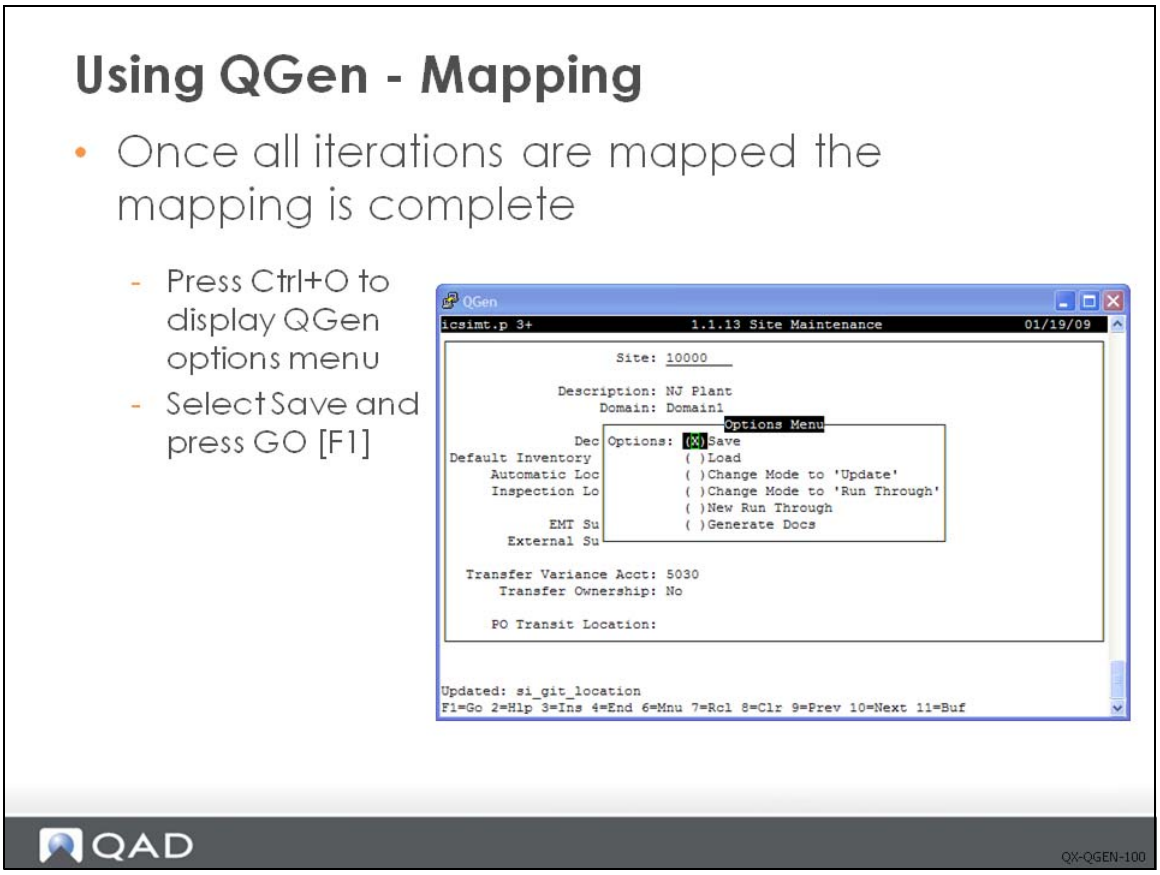
Using QGen - Mapping

- Once all paths and fields are mapped, exit the iteration

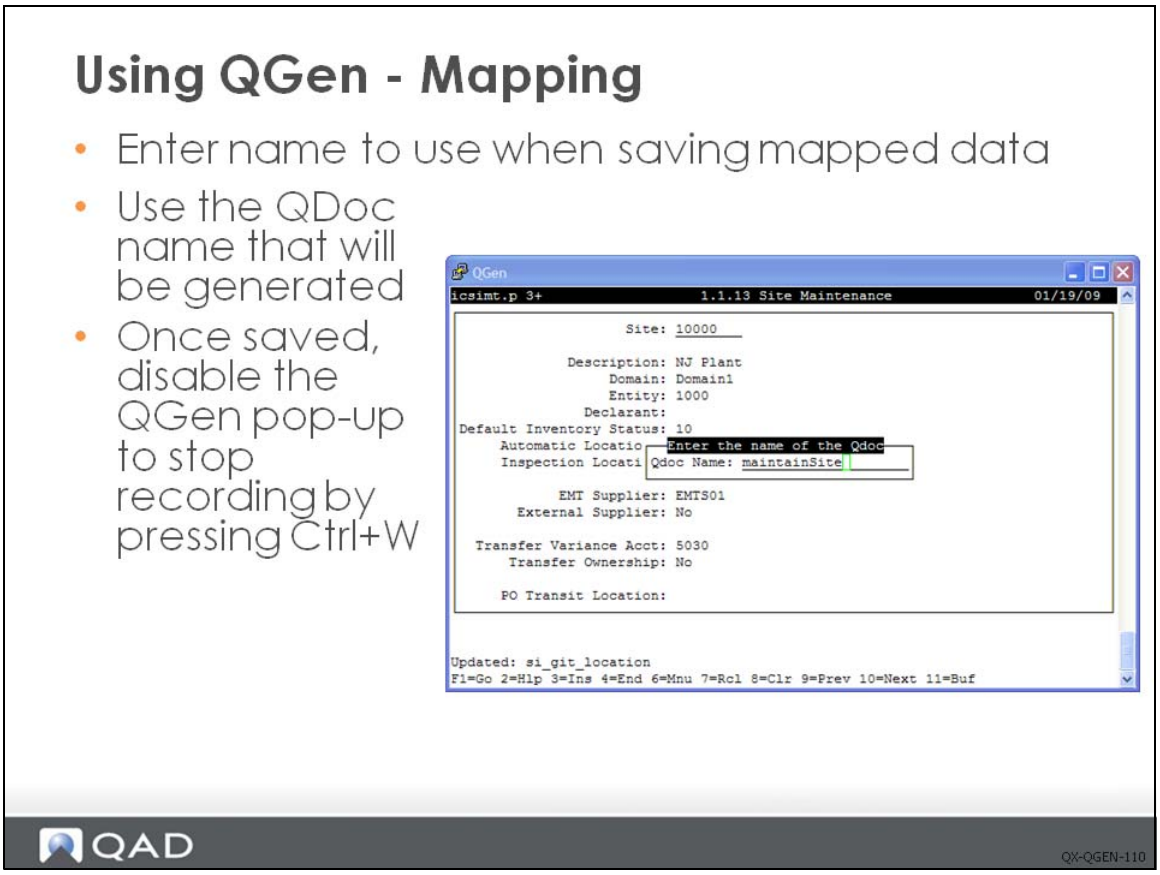


QAD QX-QGEN-090

If the Continue option is not selected, a message displays instructing the user to exit the iteration. The user is returned to the iteration and should then exit.



Once you have finished mapping the function, save the data that has been mapped. Press Ctrl+O to display the QGen options menu. Select Save and press Go.

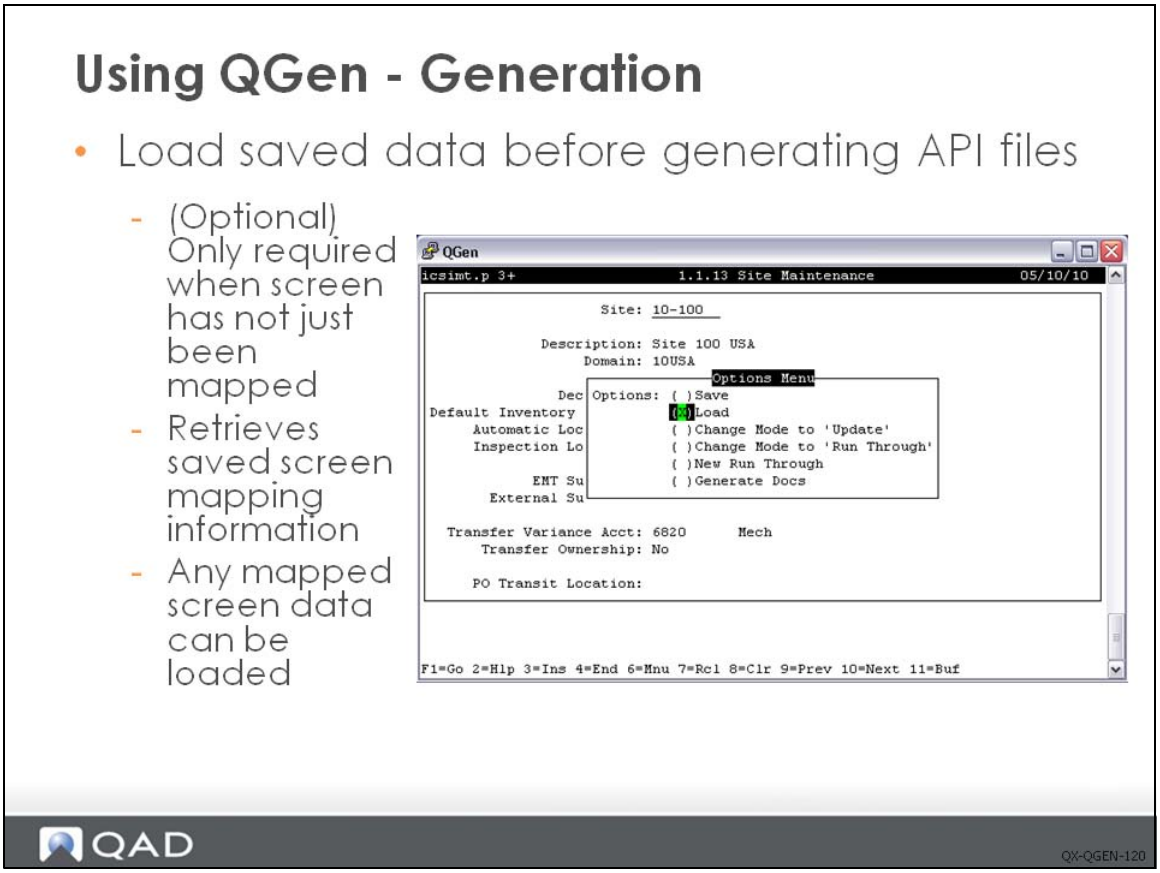


The system prompts you for a file name to save the recorded data. You should use the name of the QDoc that will be generated as this helps to identify the data file that was used to generate the QXtend artifacts for a specific QDoc.

The naming standard for QDocs is <*verb*><*object*>. The verb is the action performed by the API; for example, maintain, query, update, and so on. The object is the application component being acted on; for example, SalesOrder, PurchaseOrder, Customer, and so on.

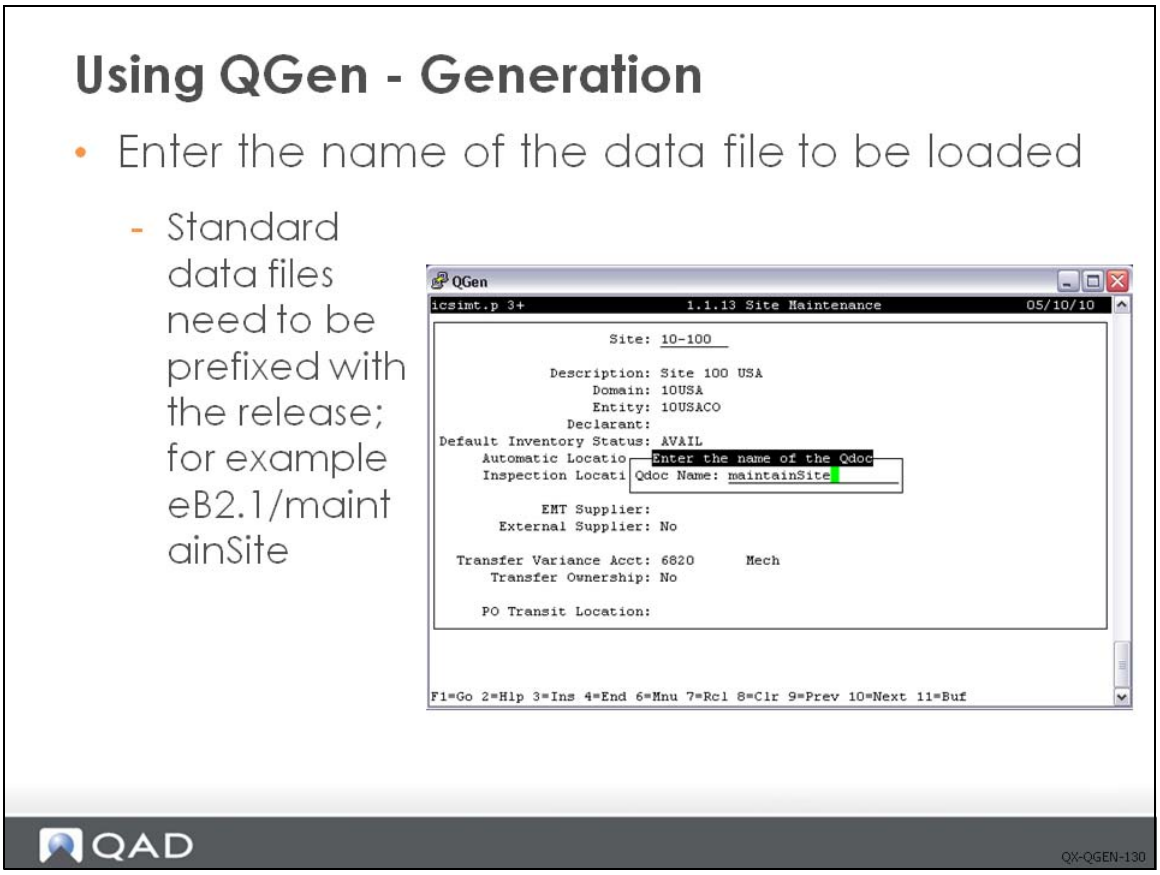
Once saved, disable the QGen pop-up to stop recording by pressing Ctrl+W.

Generating API Files



Before generating the API files, reload the data that was saved at the end of the mapping of the screen. This step is optional if you have just completed the mapping of the screen that you want to generate the API for.

You can use the Load option to load the information recorded for a screened map from the mapping file. API files can be generated/re-generated anytime: you do not have to generate them immediately after mapping the screen. Loading the API data is also useful when mapping a customized screen as only the customized fields will need to be mapped if you use the original QGen data file.



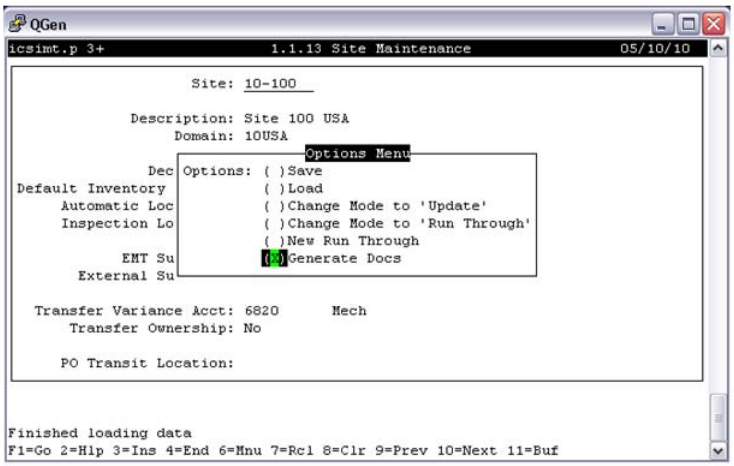
Selecting the Load option displays a system prompt for the name of the mapping data file that you want to load. If you are loading a data file you created, type in the name of the file you used to save the mapping data.

If you want to load a data mapping file that was shipped with QXtend, you must prefix the file name with the release name as used in QXtend. The standard release names are:

- eB
- eB2
- eB2.1
- QADSE
- QADEE

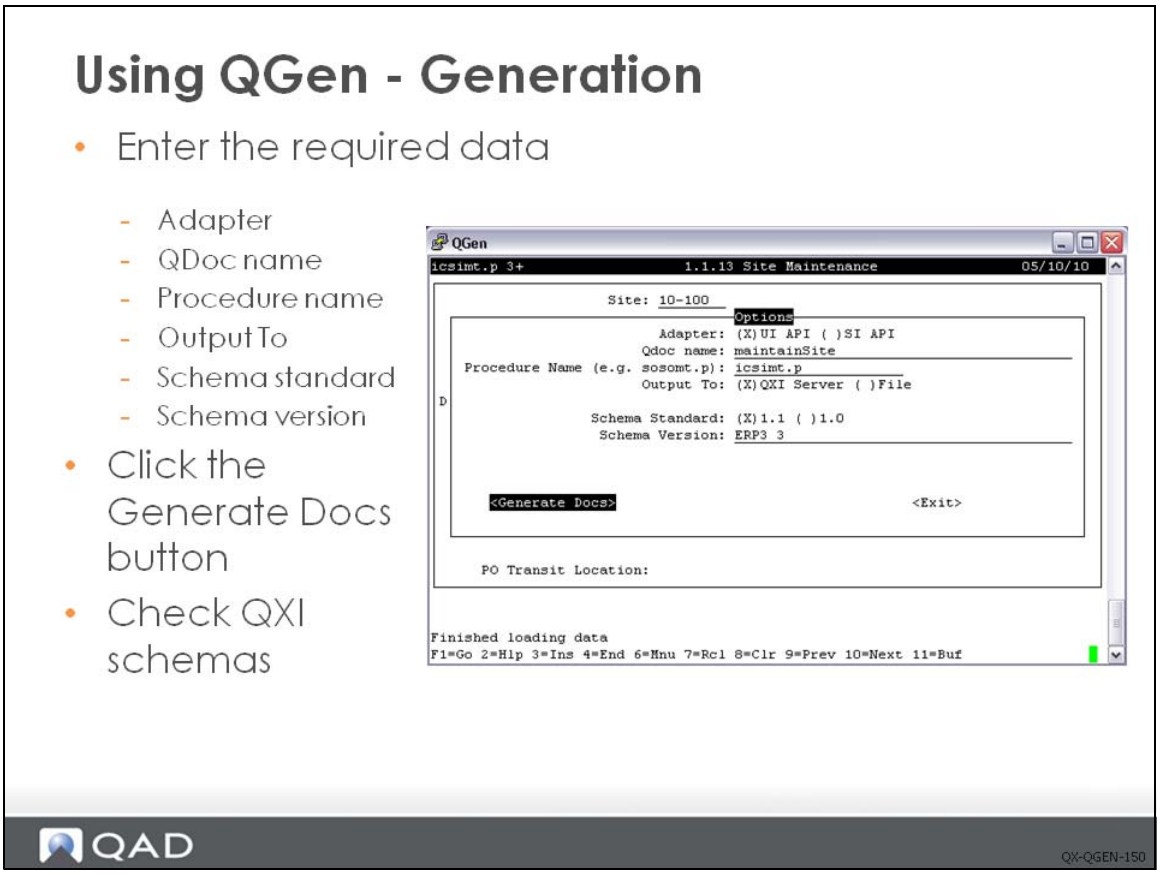
Using QGen - Generation

- Open the QGen menu (Ctrl+O)
  - Select the Generate Docs menu option



QAD

QX-QGEN-140



QGen requires the following additional information to generate the artifacts required by QXtend for the UI API adapter:

Adapter. Choose UI API adapter. The SI API option is used for generate customized schemas for native API. For details on generating native APIs, see Technical Reference: *QAD QXtend*

Qdoc Name. Name of the QDoc to be generated. The name should follow the usual QDoc naming guidelines using the <*verb*><*object*> format; for example, maintainSalesOrder.

Procedure Name. The name of the procedure the API has been mapped from. This is used to generate the name of the events file; for example, `icismt.p` generates an events file called icsimt-ERP3_1.xml.

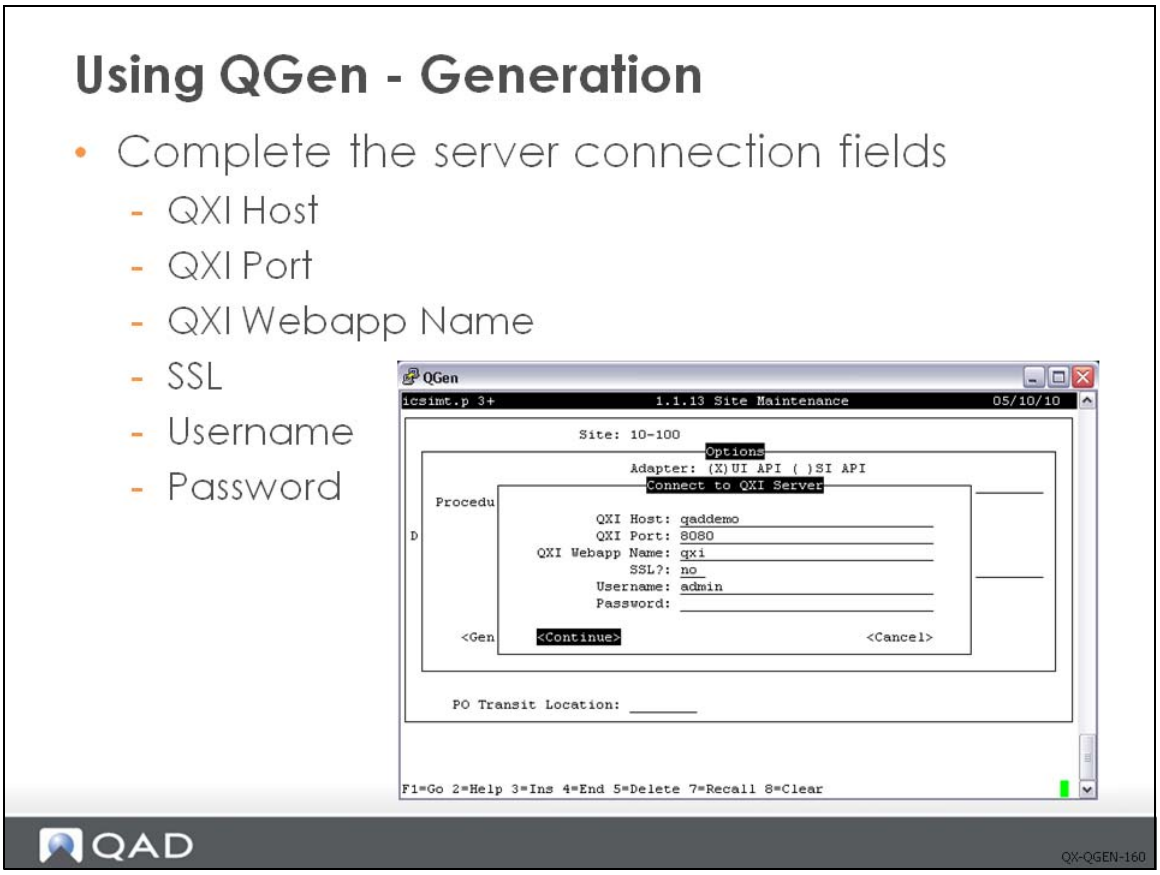
Output To. Deploy to QXI Server or output files to the local server. Typically the file generated by QGen is saved to the QXtend adapter installation directory: `<qxi-adapter-home>\tools\generatedDocs`

Schema Standard. New API's should use the default (QDoc 1.1). The 1.0 standard is provided for backward compatibility only.

Schema Version. The version number to assign to the schema and the API.

QGen generates the following files:

- <*program*>-<*version*>.xml – the events file
- <*qdoc-name*>-<*version*>.xsd – the request XML Schema file
- <*qdoc-name*>Response-<*version*>.xsd – the response XML Schema file



To deploy API to QXI, QGen needs to know how to connect to QXI web service. This screen is used to complete the server connection fields.

Note The information in this screen persists in memory until QGen is closed.

QXI Host. Specify the hostname of the QXI server.

QXI Port. Specify the port number used to access the QXI server.

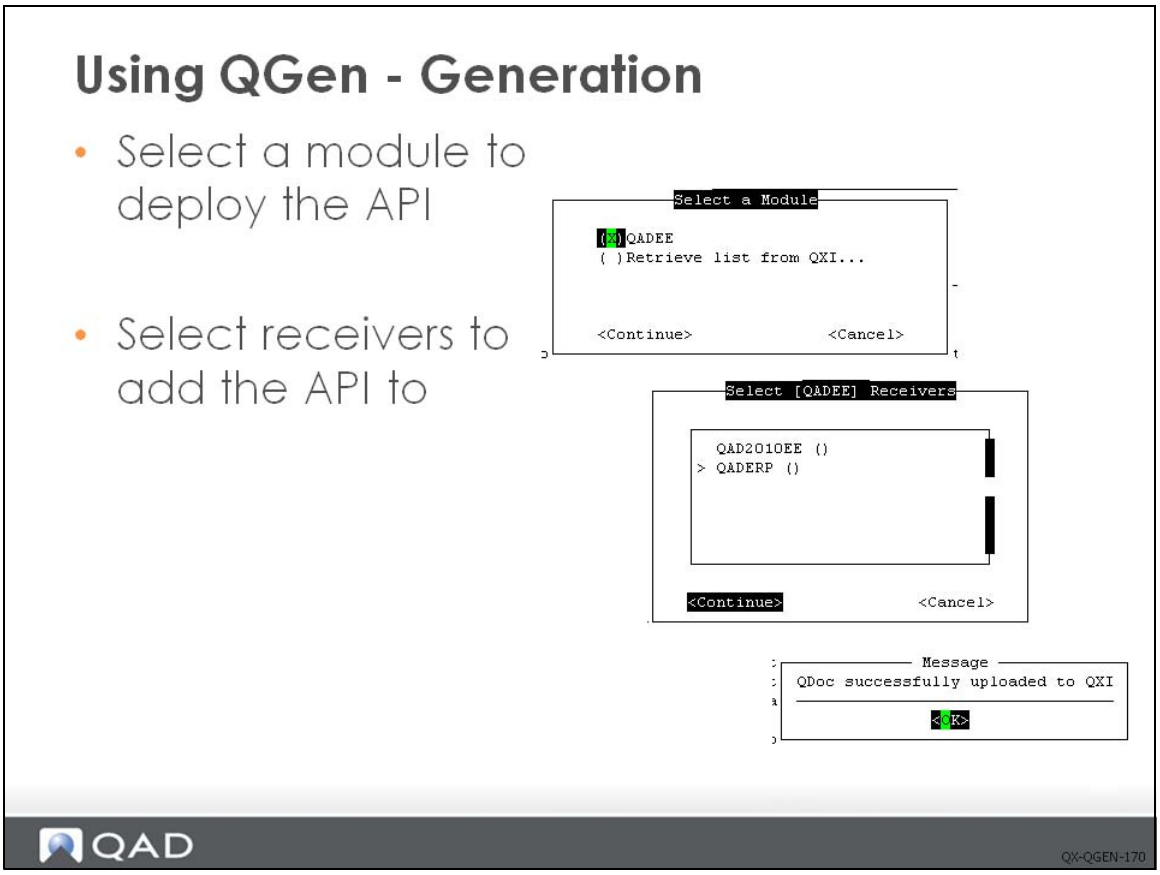
QXI Webapp Name. Specify the name of the Web service used to connect to the QXI server.

SSL. Select the field to use Secure Socket Protocol https encryption on messages to the QXI server.

Note If SSL is selected, then QXI Host must include domain name of the host.

Username. Specify the username login for the QXI server.

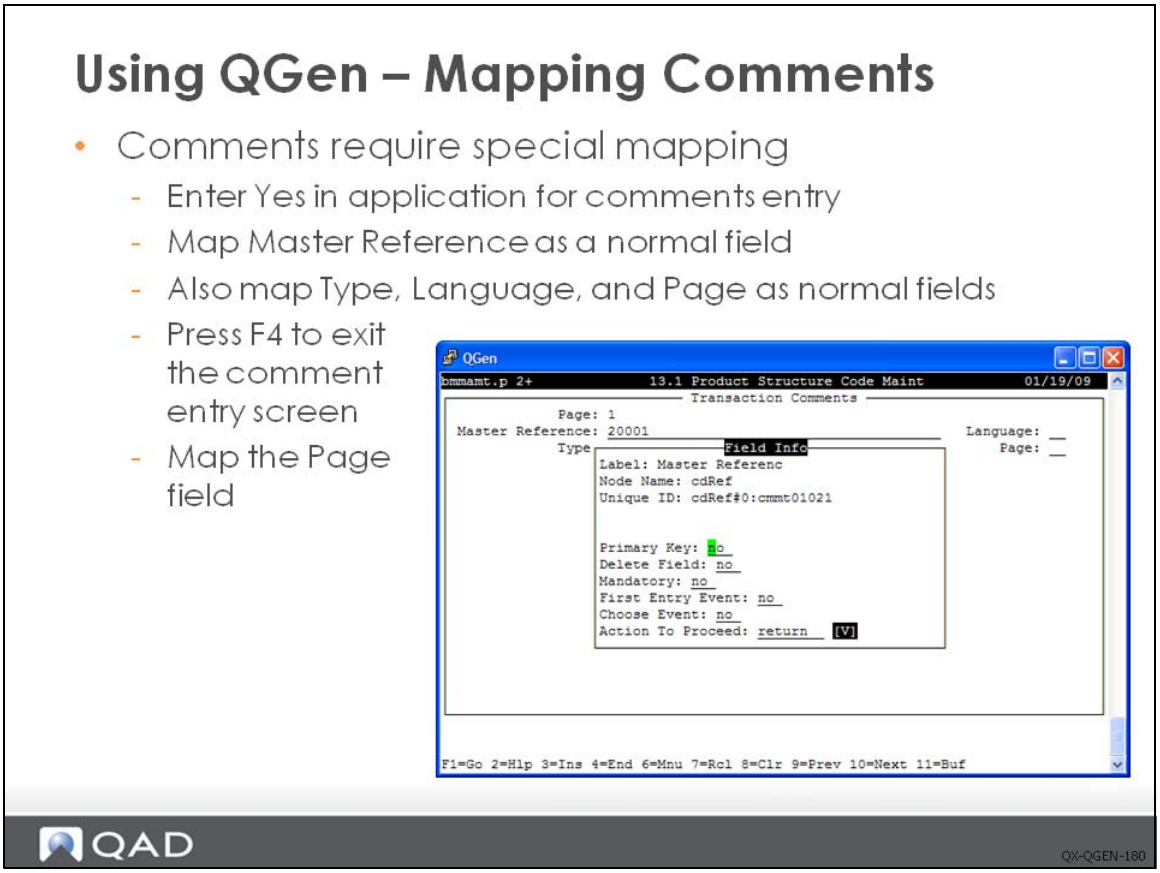
Password. Specify the password for the user.



Select the module to which you want to send the QDocs and then select Continue. The Select Receivers screen displays.

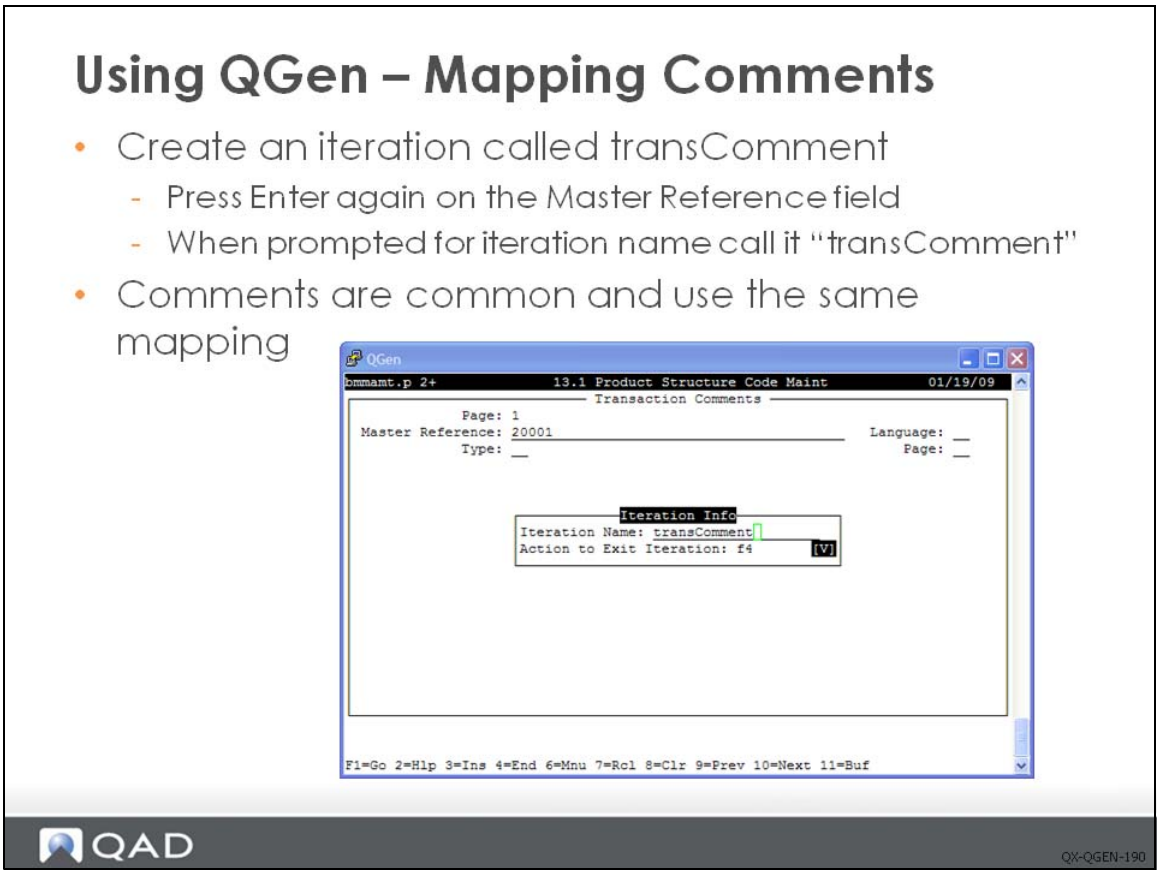
Then, select the receiver to which you want to send the QDocs and then select Continue. If the upload of QDocs is successful, the Upload QDoc to QXI Successful message displays.

Mapping Comments



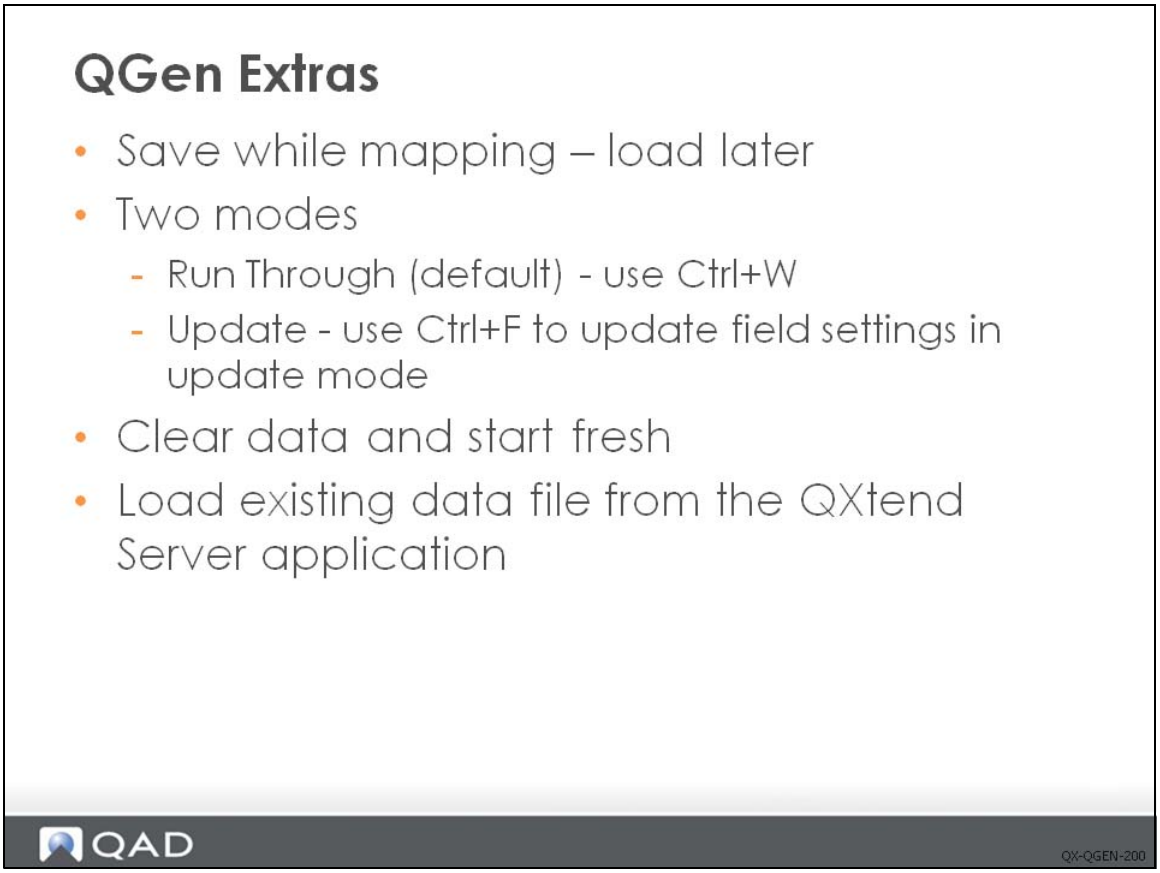
Mapping functions that contain transaction comments requires special mapping. The structure for comments is common across the entire application, so when mapping a screen that includes transaction comments, you should re-use the screen mapping that has already been performed.

When mapping a function with transaction comments, map the screen as normal. Set the comments flag to Yes to ensure that the comments entry screen displays. Map the first four or five fields on the comments as usual. When the system prompts you to enter the 15 lines of comment details for that page, press F4 to exit the update screen. The system prompts you for the page. Press Enter and update the values. The system returns you to the Master Reference field. Press Enter again so that QGen recognizes the iteration.



QGen prompts you for the iteration name for the transaction comments transaction you are mapping. Use the iteration name `transComment`. QGen will recognize this value and substitute the premapped schema definition for transaction comments.

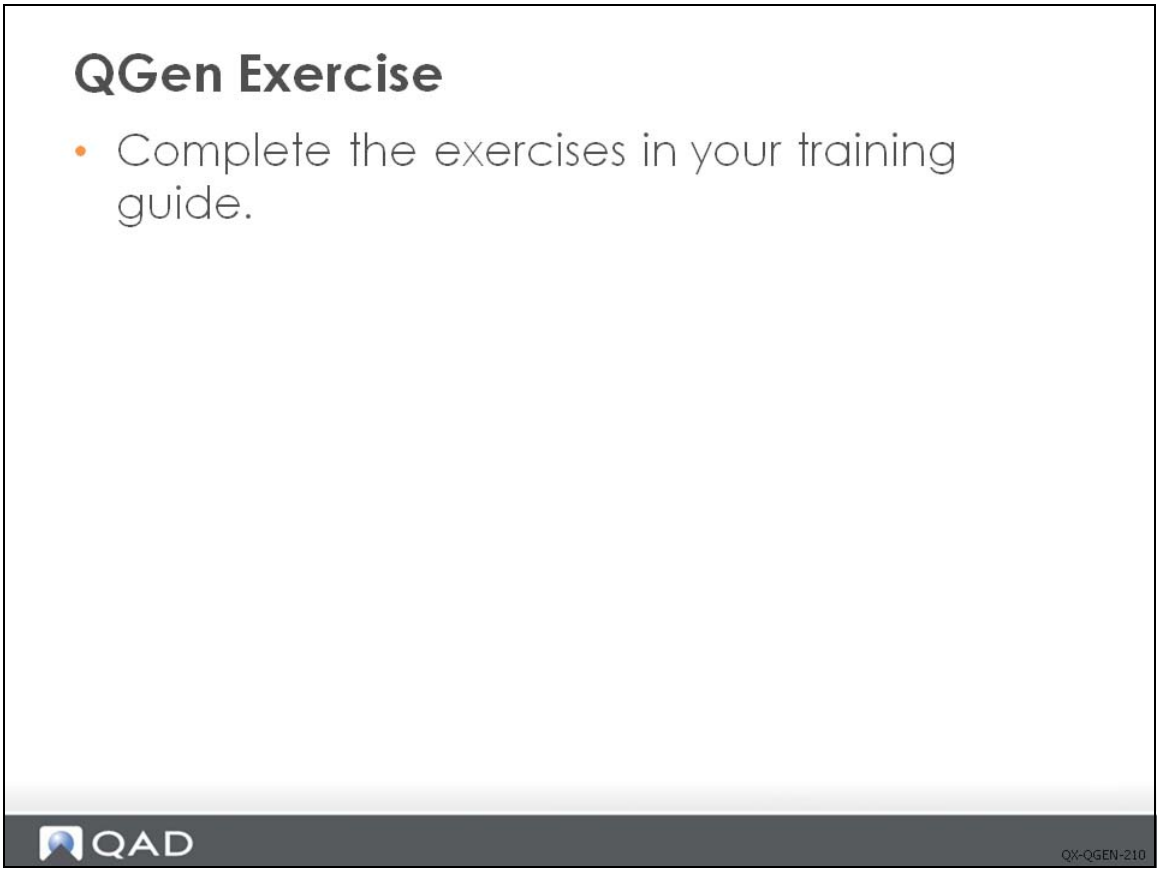
QGen Extras



QGen has several other features that have not been covered:

- You can save your work to a data file if you are interrupted and come back to it later simply by saving and then reloading and carrying on later. All of the fields already mapped will not prompt for information again.
- There are three operating mode available:
  - Run Through: The default mode records information field by field as it is accessed.
  - Update: Allows the user to change already recorded info. Ctrl+F is used to access the data already recorded.
  - New Run Through: Clears all of the data that has been recorded and allows the user to start from scratch.
- Mapping data files are shipped with the QXtend server so these files can be loaded when mapping a slightly modified function. The standard file can be used as a base and the new data can be added to the data file. This saves a lot of time as the standard data does not have to be recorded again only the customizations need to be recorded.
- There is a QGen Help.doc available on the CD, which is normally installed along with the QGen source code on the QAD EA machine.

Exercise: QGen



The following list shows a number of key concepts used in QGen in QXI. In each statement below, fill in the correct term from the list.

F1	twice
Ctrl+O	QDoc
structure and field	QXI server
Ctrl+W	Field Info
schema	receivers
module	

1. QGen is a tool that records the _____ information from QAD Enterprise Applications character user interface screens and saves it to data files.
2. The data recorded by the mapping process is passed to the QGen generation engine, which transforms the data in the files into the relevant QDoc _____ files and events.
3. To enable the QGen mapper, start a QAD EA session, select the function you want to map, and then press _____to enable the mapper.
4. You enter the information about the field you want to record into the _____ pop-up window.
5. Once you've entered all the information about a field in the Field Info pop-up window, press _____ to move to the next field.

6 Where there are frames that you enter iterations, you must press Enter _____ to ensure that the mapper picks up the iteration name.

7 After mapping the UI function, press _____ to display the Options menu. Select Save on the menu and then press Go.

8 When entering the name of the file in which to save the mapping data, it's recommended that you use the same name as the _____ that will be generated.

9 Deploying an API usually includes following steps: enter required data, connect to _____, select a _____ and select _____.

Lab: QGen

QAD QXtend 1.7.2 has been installed with both QXI and QXO components. In the previous labs you performed the basic configuration that is required to enable requests to be processed by QXI and QXO. The following lab exercises describe how to use QGen to map screens in the Enterprise Applications and create UI API adapter APIs.

`<LabHomeDirectory> = C:\QXtendTraining\Labs\05-QGen\`

QAD QXtend Inbound ships with a limited set of APIs; most of them use the UI API adapter. However, many screens in QAD Enterprise Applications have not been mapped; you can generate UI API adapter APIs for these screens using QGen. You can also use QGen to map screens that have been created or modified during an implementation project.

1. Mapping Simple Screens

The QGen tool is installed during the installation of QAD QXtend, but the script to access QGen is not automatically created. The QGen script runs QAD Enterprise Applications in standalone character mode and must be launched from a standard telnet session. QGen cannot be run from within the QAD .Net UI.

To access QGen from the training environment:

1. Open PUTTY use the icon on the training Windows machine Desktop.



2. Double-click the `qaddemo` saved session.
3. Log in as user `demo-admin`, password `qad`.
4. Run the QGen script by entering:
   `/dr01/qadapps/qea/qxtend/scripts/client.qgen`
5. Log in using the user ID `qmi` and a blank password.

1.1 Map the Generalized Code Maintenance Screen

Mapping a screen in QAD Enterprise Applications requires you to walk through each of the fields/frames in a function and record the actions taken by the user to enter data. The first part of this exercise describes mapping a simple application screen: Generalized Code Maintenance.

In your telnet session, do the following:

1. Launch the Generalized Codes Maintenance function (36.2.13).
2. Press Ctrl+W to enable the QGen auto pop-up functionality that automatically prompts you for details about each field.
3. Press Ctrl+O to access the QGen menu.

4 Select the New Run Through mode by pressing the spacebar. (Move down the options by using the down arrow key.)

5 Press F1.

6 When the system asks if you want to reset, acknowledge the reset action by pressing the spacebar. Then press F1.

7 Walk through the Generalized Codes Maintenance screen, while updating an existing entry:

   a UI Field = Field Name.

   Enter `abd_conv` and press Enter.

   In the Field Info screen, set Primary Key to Yes, and Delete Field to No.

   Leave all other fields as their default values.

   Press F1 to continue.

   b UI Field = Value.

   Press Enter.

   In the Field Info screen, set Primary Key to Yes, and Delete Field to No.

   Leave all other fields as their default values.

   Press F1 to continue.

   c UI Field = Comments.

   Press Enter.

   In the Field Info screen set Primary Key to No, and Delete Field to Yes.

   Leave all other fields as their default values.

   Press F1 to continue.

   d You are returned to the Field Name field. For QGen to recognize an iteration, it has to identify that it has encountered a field twice.

   Enter `abd_conv` and press Enter.

   In the Iteration Info frame, enter the name to use for the iteration.

   Iteration Name = `generalizedCode`

   Action to Exit = F4.

   Press F1 to continue.

   Choose not to continue with the iteration

8 Save the information recorded while mapping the Generalized Code Maintenance screen:

   a Press Ctrl+W to disable the QGen automatic pop-up.

   b Press Ctrl+O to access the QGen menu.

   c Select Save and press F1.

   d When prompted for the name of the QDoc, enter:

   `maintainGeneralizedCode`

9 Load the information record while mapping the Generalized Code Maintenance screen:

a Press Ctrl+O to access the QGen menu.

b Select Load and press F1.

c When prompted for the name of the QDoc, enter:
`maintainGeneralizedCode`

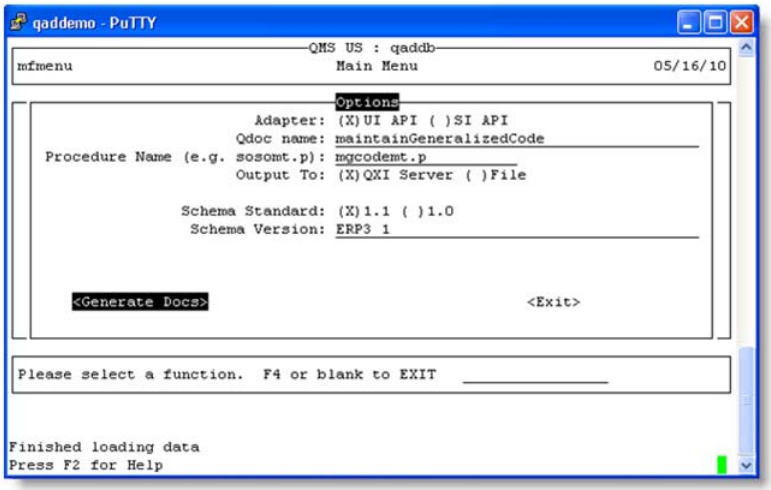
10 Generate the API files that are required by QXI:

a Press Ctrl+O to access the QGen menu.

b Select Generate Docs and press F1.

c Set the following values:

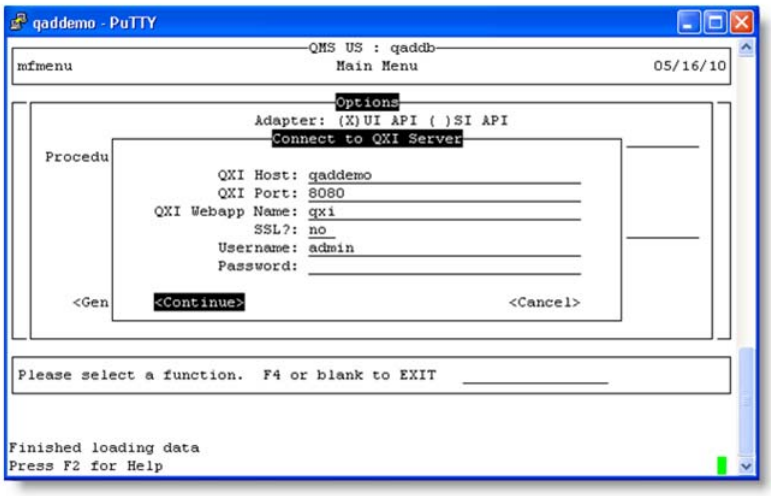
Adapter	UI API
QDoc name	maintainGeneralizedCode
Procedure Name	mgcodemt.p
Output To	QXI Server
Schema Standard	1.1 (Press Tab to tab to the Schema Version field.)
Schema Version	ERP3_1



d Select the Generate Docs option and press Enter.

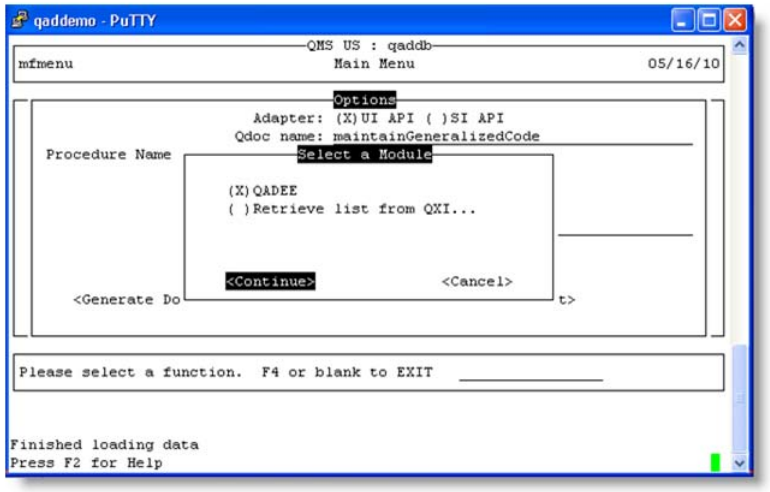
e Set the following values in Connect to QXI Server screen

QXI Host	qaddemo
QXI Port	8080
QXI Webapp Name	qxi
SSL	no
Username	admin
Password	mfgpro

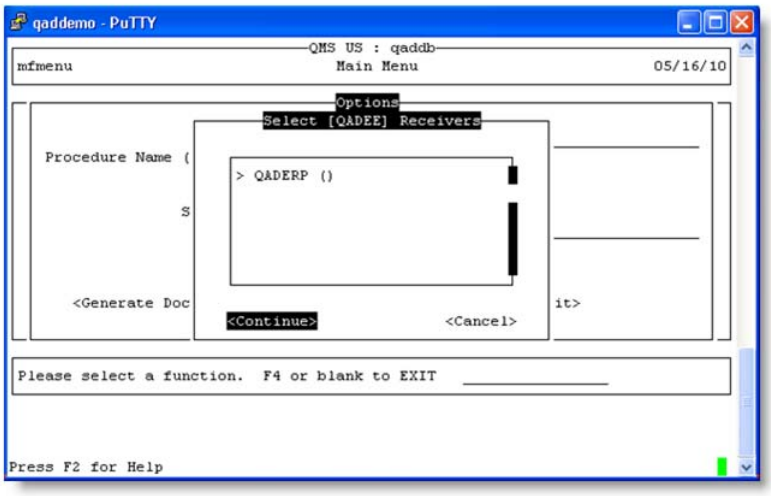


f Press Continue

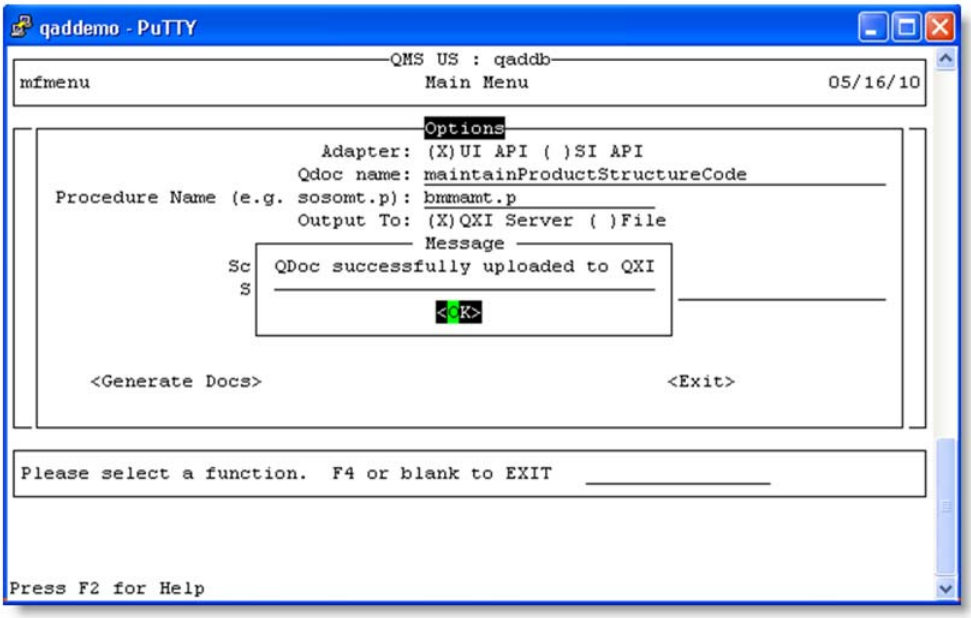
g Select QADEE and press Continue



h Choose QADERP and press Continue



i Press OK if QDoc successfully uploaded to QXI



j Press F4 to exit the Generate Docs screen.

1.2 Verify the Generalized Code API has been added

To verify that the maintainGeneralizedCode API was successfully added to QXI, you need to check two places. First check that the API is available as a custom API for the QADEE application type; second, check it is a supported API on the QADERP receiver.

1 Open the QXI Web application in Internet Explorer:

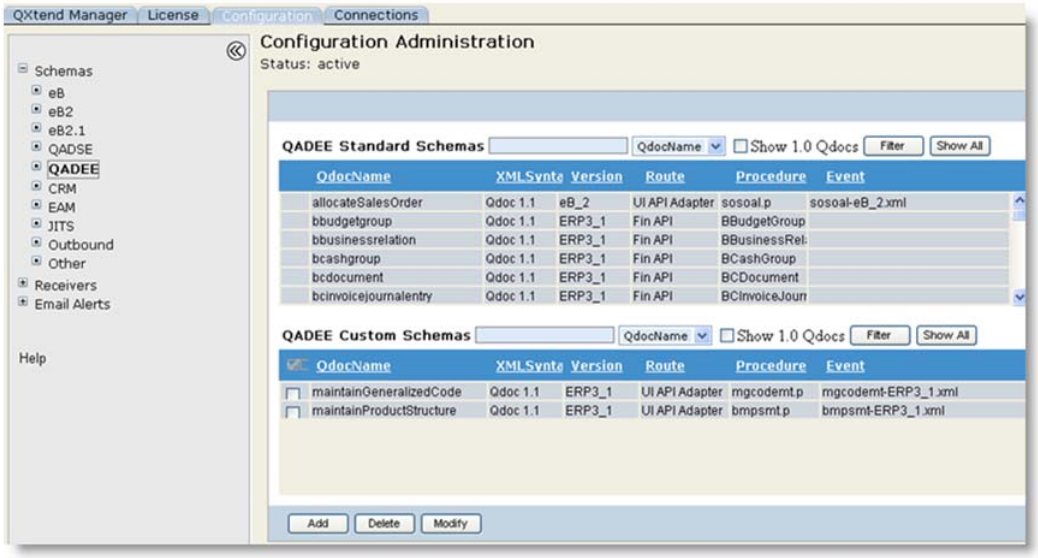
`http://qaddemo:8080/qxi`

2 Select the Configuration tab.

3 Select the Schemas node on the menu tree.

4 Select QADEE.

The maintainGeneralizedCode version ERP3_1 should display as the custom API that is available for the QADEE application type as shown here:



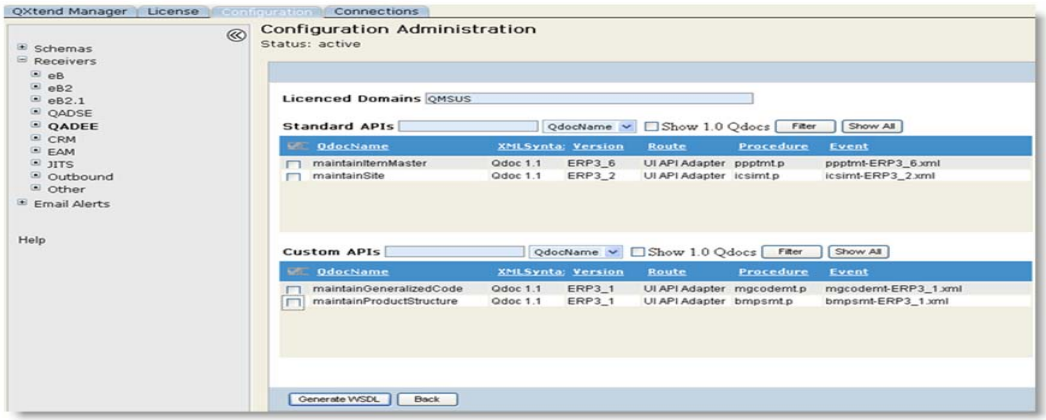
5 Select the Configuration tab.

6 Select the Receivers node on the menu tree.

7 Select QADEE.

8 Select the check box next to the QADERP receiver and click the View button.

The maintainGeneralizedCode API version ERP3_1 should display as a supported QDoc for the receiver as shown here:



1.3 Test the New API

The new version of the Generalized Code Maintenance API has now been successfully deployed to QXI. However, we will have to use the new version to process a few transactions before we can be sure everything has been uploaded correctly.

The best way to test a QAD QXtend API is by using soapUI to build a WSDL project, create some requests, and process those requests using soapUI.

Generate the WSDL file first.

1 Open the QXI Web application in Internet Explorer:

`http://qaddemo:8080/qxi`

2 Select the Configuration tab.

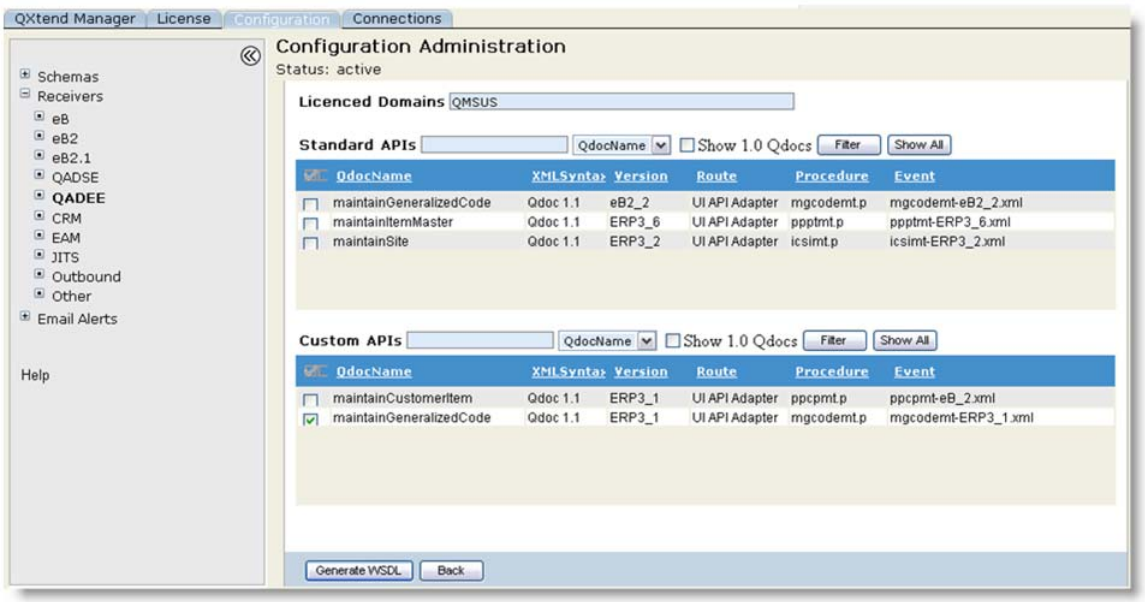
3 Select the Receivers node on the menu tree.

4 Select QADEE from the list of receivers.

5 Select the check box next to the QADERP receiver.

6 Click the View button.

7 Select maintainGeneralizedCode from the list of Custom APIS.



8 Click the Generate WSDL button.

9 Click the No button for “Create backward compatible WSDL?”

10 Save maintainGeneralizedCode-ERP3_1.zip to <*LabHomeDirectory*>

11 Unzip the file to <*LabHomeDirectory*>/`maintainGeneralizedCode`.

12 Open soapUI on the Windows image. Use the shortcut on the Desktop.

13 Create a new workspace.

a File – New Workspace.

b Set the workspace name to Lab 05 – QGen.

c Save the workspace file in <*LabHomeDirectory*>.

14 Create a new WSDL project.

a Right-click on the workspace name.

b Select the New soapUI Project option.

c Set the project name to QGen 05.

d Set the WSDL file to:

<*LabHomeDirectory*>/maintainGeneralizedCode/maintainGeneralizedCode-ERP3_1.wsdl

e Click OK to create the project.

15 Drill down to the Request 1 message created under the QGen 05 project.

16 Right-click Request 1 and select the Clone Request option. Enter a name for the new request.

17 Edit the new request so that it can be processed by QXI. Change the SOAP header:

a Set the receiver to `QADERP`.

b Set `suppressResponseDetail` to `true`.

18 Create the necessary session context entries. Create two ttContext iterations:

a Qualifier = `QAD`, Name = `version`, Value = `ERP3_1`

b Qualifier = `QAD`, Name = `domain`, Value = `11CAN`

Note If you copy and paste the empty `ttContext` node and blank lines are created, right-click the request and select the Format XML option. soapUI will remove any blank lines and correct the indentation of the XML.

19 Edit the application data section of the message:

a The dsGeneralizedCode node by default will contain all of the fields available when processing a Generalized Code update. However, there are only a few fields that are absolutely required for a transaction to process successfully, the best way to identify the mandatory fields is to run through the UI and see which fields you have to enter data for.

b Edit the XML and create a new Generalized Code QDoc that will be processed successfully by QAD QXtend Inbound, use the following values:

codeFldname	qxtest01
codeValue	test
codeCmmt	Qxtend Training Test

20 Process the QDoc message that you have created in the new soapUI project and verify that the Generalized Code is created successfully.

21 Create and process a new message that will delete the Generalized Code created in the previous step.

22 Verify the changes in QAD EE.

2. Mapping Complex Screens: Parent-Child Iterations

A common structure that is used across the QAD Enterprise Application suite is the parent-child data structure. Some good examples of these interfaces would be Purchase Order, Sales Order, and so on. When mapping a screen that has multiple iterations (repeating parent-child relationships), it is important to remember that you have to enter two lots of data into the iteration for QGen to recognize the iteration.

2.1 Map the Product Structure Maintenance Screen

This part of the exercise describes mapping a complex application screen: Product Structure Maintenance.

In the telnet session opened above, do the following:

1 Launch the Product Structure Maintenance function (13.5).

2 Press Ctrl+W to enable the QGen auto pop-up window.

3 Press Ctrl+O to access the QGen menu.

4 Select the New Run Through mode.

5 Press F1.

6 When asked if you want to reset, acknowledge the reset action.

7 Walk through the Product Structure Maintenance screen, while updating an existing entry:

   a UI Field = Parent Item.

   Enter `01010` and press Enter.

   In the Field Info screen set the Primary Key field to Yes, and Delete Field to No.

   Leave all other fields as their default values.

   Press F1 to continue.

   b UI Field = Component Item.

   Enter `50001` and press Enter.

   In the Field Info screen set Primary Key to Yes, and Delete Field to No.

   Leave all other fields as their default values.

   c Repeat this process for the Reference and Start Effective fields, but do not enter any data—just press Enter and complete the Field Info frame using the same Primary Key and Delete Field settings as above.

   d Map the remaining fields by accepting the data that is stored in the application; there is no need to change the data. Make sure that you set the Primary Key to No and the Delete Field to Yes.

8 When you are taken back to the Component Item field, press Enter again (if you do not press Enter again, QGen will not record the iteration).

   a In the Iteration Info frame set the Iteration Name to `componentItem`, and Action to Exit to F4.

9 Do not continue with the iteration.

10 Press F4 once to exit the componentItem iteration.

11 On the Parent Item field, remember to enter into the maintenance of the Product Structure again to ensure that QGen maps the iteration. Press Enter.

   a In the Iteration Info frame, set Iteration Name to `productStructure` and Action to Exit to F4.

12 Do not continue with the iteration.

Save the information recorded while mapping the Product Structure Maintenance screen:

1 Press Ctrl+W to disable the QGen automatic pop-up.

2 Press Ctrl+O to access the QGen menu.

3 Select Save and press F1.

4 When prompted for the name of the QDoc, enter `maintainProductStructure`.

Load the information record while mapping the Product Structure Maintenance screen:

1 Press Ctrl+O to access the QGen menu.

2 Select Load and press F1.

3 When prompted for the name of the QDoc, enter `maintainProductStructure`.

Generate the API files that are required by QXI:

1 Press Ctrl+O to access the QGen menu.

2 Select Generate Docs and press F1.

3 Set the following values:

Adapter	UI API
QDoc name	maintainProductStructure
Procedure Name	bmpsmt.p
Output To	QXI Server
Schema Standard	1.1
Schema Version	ERP3_1

4 Click the Generate Docs button and press Enter.

5 Set the following values in Connect to QXI Server screen

QXI Host	qaddemo
QXI Port	8080
QXI Webapp Name	qxi
SSL	no
Username	admin
Password	mfgpro

6 Press Continue.

7 Select QADEE and press Continue.

8 Choose QADERP and press Continue.

9 Press OK if QDoc successfully uploaded to QXI.

10 Press F4 to exit the Generate Docs screen.

2.2 Verify the maintainProductStructure API has been added

To verify that the maintainGeneralizedCode API was successfully added to QXI, you need to check two places. First check that the API is available as a custom API for the QADEE application type; second, check it is a supported API on the QADERP receiver

1 Open the QXI Web application in Internet Explorer:

```
http://qaddemo:8080/qxi
```

2 Select the Configuration tab.

3 Select the Schemas node on the menu tree.

4 Select QADEE.

The maintainProductStructure version ERP3_1 should display as the custom API that is available for the QADEE application type.

5 Select the Configuration tab.

6 Select the Receivers node on the menu tree.

7 Select QADEE.

8 Select the check box next to the QADERP receiver and click the View button.

The maintainProductStructure API version ERP3_1 should display as a supported QDoc for the receiver.

2.3 Test the New API

The new version of the Product Structure Maintenance API has now been successfully deployed to QXI. However, until the new version has been used to process a couple of transactions, you cannot be sure that everything has been uploaded correctly.

The best way to test a QAD QXtend API is by using soapUI to build a WSDL project, create some requests, and process those requests using soapUI.

Generate the WSDL file first.

1. Open the QXI Web application in Internet Explorer:
   `http://qaddemo:8080/qxi`
2. Select the Configuration tab.
3. Select the Receivers node on the menu tree.
4. Select QADEE from the list of receivers.
5. Select the check box next to the QADERP receiver.
6. Click the View button.
7. Select maintainProductStructure from the list of Custom APIS.
8. Click the Generate WSDL button.
9. Click the No button for “Create backward compatible WSDL?”
10. Save maintainProductStructure-ERP3_1.zip to <*LabHomeDirectory*>.
11. Unzip the file to <*LabHomeDirectory*>/`maintainProductStructure`.
12. Open soapUI on the windows image; use the shortcut on the Desktop.
13. Open the Lab 05 – QGen Workspace.
14. Create a new WSDL Project.
    a. Right-click the workspace name.
    b. Select the New soapUI Project.
    c. Set the project name to QGen 05 – Product Structure.
    d. Set the WSDL file to:
       <*LabHomeDirectory*>`/maintainProductStructure/maintainProductStructure-ERP3_1.wsdl`
    e. Click OK to create the project.
15. Drill down to the Request 1 message created under the QGen 05 – Product Structure project.
16. Right-click Request 1 and select the Clone Request option. Enter a name for the new request.
17. Edit the new request so that it can be processed by QXI.
18. Change the SOAP header:
    - Set the receiver to be QADERP.
    - Set `suppressResponseDetail` to `true`.
19. Create the necessary session context entries:
    a. Create two `ttContext` iterations.
    b. Qualifier = `QAD`, Name = `version`, Value = `ERP3_1`.

c Qualifier = `QAD`, Name = `domain`, Value = `11CAN`.

Note If you copy and paste the empty `ttContext` node and blank lines are created, right-click the request and select the Format XML option. soapUI will remove any blank lines and correct the indentation of the XML.

20 Edit the application data section of the message:

a The `dsProductStructure` node by default will contain all the fields available when processing a Product Structure update. Only a few fields are required for a transaction to process successfully; the best way to identify the mandatory fields is to run through the UI and see which fields you have to enter data for.

b Edit the XML and create a new Product Structure QDoc that will update the end effective date of a component, use the following values:

operation	M
productStructure.psPar	01010
componentItem.psComp	50001
componentItem.psend	<today> format - YYYY-MM-DD e.g. 2009-02-09

21 Process the QDoc message that you have created in the new soapUI project and verify that the Product Structure Code is updated successfully.

3. Mapping Complex Screens: Comments

Many functions in QAD Enterprise Applications allow you to create general comments that are assigned to the data you are maintaining. The structure and UI for maintaining those comments is common to all components, and consequently require a special process when mapping a screen with QGen.

3.1 Map the Product Structure Code Maintenance Screen

This part of the exercise will focus on mapping a screen in the application that allows comments to be entered: Product Structure Code Maintenance.

In the telnet session opened above perform the following steps:

1 Launch the Product Structure Code Maintenance function (13.1).

2 Press Ctrl+W to enable the QGen auto pop-up window.

3 Press Ctrl+O to access the QGen menu.

4 Select the New Run Through mode.

5 Press F1.

6 When asked if you want to reset, acknowledge the reset action.

7 Walk through the product structure code maintenance screen, while updating an existing entry:

a UI Field = BOM Code.

Enter `01010` and press Enter.

In the Field Info screen set Primary Key to Yes and Delete Field to No.

Leave all other fields as their default values.

Press F1 to continue.

b UI Field = Description.

Keep the value and press Enter.

In the Field Info screen set Primary Key to No and Delete Field to Yes.

Leave all other fields as their default values.

Press F1 to continue.

c UI Field = Unit Of Measure.

Press Enter.

In the Field Info screen set Primary Key to No and Delete Field to Yes.

Leave all other fields as their default values.

Press F1 to continue.

d UI Field = Comments.

Enter `Yes`. Press Enter.

In the Field Info screen set Primary Key to No and Delete Field to Yes.

Leave all other fields as their default values.

Press F1 to continue.

8 The standard comments entry screen displays. Follow these steps to correctly map the comments (try to ensure you are adding comments to an object that does not already have comments defined):

a UI Field = Master Reference.

Press Enter.

In the Field Info screen set Primary Key to No and Delete Field to No.

Leave all other fields as their default values.

Press F1 to continue.

b UI Field = Type.

Press Enter.

In the Field Info screen set Primary Key to No and Delete Field to No.

Leave all other fields as their default values.

Press F1 to continue.

c UI Field = Language.

Press Enter.

In the Field Info screen set Primary Key to No and Delete Field to No.

Leave all other fields as their default values.

Press F1 to continue.

d UI Field = Page.

Press Enter.

In the Field Info screen set Primary Key to No and Delete Field to No.

Leave all other fields as their default values.

Press F1 to continue.

On the comment detail entry page press F4—do not enter any comments.

e UI Field - Page.

Press Enter.

In the Field Info screen set Primary Key to No and Delete Field to No.

Leave all other fields as their default values.

Press F1 to continue.

f When you are taken back to the Master Reference field, remember to press Enter again (if we do not press Enter again QGen will not record the iteration). The comments iteration are always called `transComment`.

In the Iteration Info frame set the following values:

Iteration Name = `transComment`

Action to Exit = f4.

Do not continue with the iteration.

Exit the comments screen.

g When you are taken back to the BOM Code field, remember to press Enter again (if we do not hit enter again QGen will not record the iteration).

In the Iteration Info frame set the following values:

Iteration Name = `productStructureCode`

Action to Exit = f4.

Do not continue with the iteration.

Press F4 once to exit the productStructureCode iteration.

Save the information recorded while mapping the Product Structure Code Maintenance screen:

1 Press Ctrl+W to disable the QGen automatic pop-up.

2 Press Ctrl+O to access the QGen menu.

3 Click Save and press F1.

4 When prompted for the name of the QDoc, enter `maintainProductStructureCode`.

Load the information record while mapping the Product Structure Code Maintenance screen:

1 Press Ctrl+O to access the QGen menu.

2 Click Load and press F1.

3 When prompted for the name of the QDoc, enter `maintainProductStructureCode`.

Generate the API files that are required by QXI:

1. Press Ctrl+O to access the QGen menu.
2. Click Generate Docs and press F1.
3. Set the following values:

Adapter	UI API
QDoc name	maintainProductStructureCode
Procedure Name	bmmamt.p
Output To	QXI Server
Schema Standard	1.1 (Press Tab to tab to the Schema Version field)
Schema Version	ERP3_1

4. Select the Generate Docs option and press Enter.
5. Set the following values in Connect to QXI Server screen

QXI Host	qaddemo
QXI Port	8080
QXI Webapp Name	qxi
SSL	no
Username	admin
Password	mfgpro

6. Press Continue.
7. Select QADEE and press Continue
8. Choose QADERP and press Continue.
9. Press OK if QDoc successfully uploaded to QXI.
10. Press F4 to exit the Generate Docs screen.

3.2 Verify the Product Structure Code API has been added

To verify that the maintainProductStructureCode API was successfully added to QXI, you need to check two places. First check that the API is available as a custom API for the QADEE application type; second, check it is a supported API on the QADERP receiver.

1. Open the QXI Web application in Internet Explorer:
   `http://qaddemo:8080/qxi`
2. Select the Configuration tab.
3. Select the Schemas node on the menu tree.
4. Select QADEE.

The maintainProductStructureCode version ERP3_1 should display as the custom API that is available for the QADEE application type.

5 Select the Configuration tab.

6 Select the Receivers node on the menu tree.

7 Select QADEE.

8 Select the check box next to the QADERP receiver and click the View button.

The maintainProductStructureCode API version ERP3_1 should display as a supported QDoc for the receiver.

3.3 Test the New API

The new version of the Product Structure Maintenance API has now been successfully deployed to QXI. However, until the new version has been used to process a couple of transactions, you cannot be sure that everything has been uploaded correctly.

The best way to test a QAD QXtend API is by using soapUI to build a WSDL project, create some requests, and process those requests using soapUI.

Generate the WSDL file first.

1 Open the QXI Web application in Internet Explorer:

`http://qaddemo:8080/qxi`

2 Select the Configuration tab.

3 Select the Receivers node on the menu tree.

4 Select QADEE from the list of receivers.

5 Select the check box next to the QADERP receiver.

6 Click the View button.

7 Select maintainProductStructureCode from the list of Custom APIS.

8 Click the Generate WSDL button.

9 Click the No button for “Create backward compatible WSDL?”

10 Save maintainProductStructureCode-ERP3_1.zip to <*LabHomeDirectory*>

11 Unzip the file to <*LabHomeDirectory*>/maintainProductStructureCode.

12 Open soapUI on the Windows image. Use the shortcut on the Desktop.

13 Open the Lab 05 – QGen Workspace:

   a File – Switch Workspace.

   b Set the workspace name to Lab 05 – QGen.

   c The workspace file is in the <LabHomeDirectory>.

14 Create a new WSDL project.

a Right-click the workspace name.

b Select the New soapUI Project.

c Set the project name to QGen 05 – Product Structure Code.

d Set the WSDL file to:

`<LabHomeDirectory>/maintainProductStructureCode/maintainProductStructureCode-ERP3_1.wsdl`

e Click OK to create the project

15 Drill down to the Request 1 message created under the QGen 05 – Product Structure Code project.

16 Right-click Request 1 and select the Clone Request option. Enter a name for the new request.

17 Edit the new request so that it can be processed by QXI. Change the SOAP header:

a Set the receiver to `QADERP`.

b Set `suppressResponseDetail` to `true`.

18 Create the necessary session context entries: Create two `ttContext` iterations.

a Qualifier = `QAD`, Name = `version`, Value = `ERP3_1`

b Qualifier = `QAD`, Name = `domain`, Value = `11CAN`

Note If you copy and paste the empty `ttContext` node and blank lines are created, right-click the request and select the Format XML option. soapUI will remove any blank lines and correct the indentation of the XML.

19 Edit the application data section of the message:

a The `dsProductStructureCode` node by default will contain all of the fields available when processing a Product Structure Code update. However, only a few fields are required for a transaction to process successfully. The best way to identify the mandatory fields is to run through the UI and see which fields you have to enter data for.

b Edit the XML and create a new Product Structure QDoc that will update the end effective date of a component, use the following values:

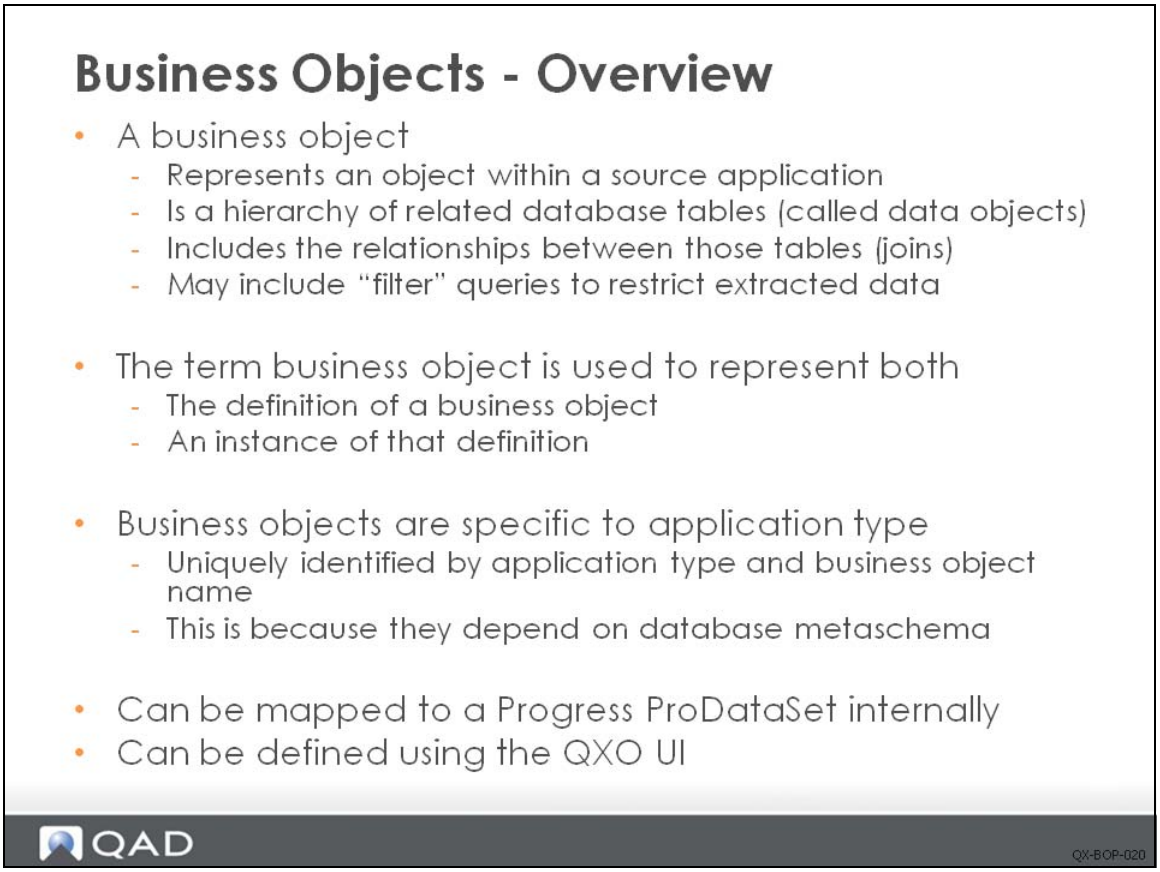
- `productStructureCode.bomParent` = `30-10000-001`
- `productStructureCode.cmmts` = `true`
- `productStructureCodeTransComment.cmmt` = `Comment Line 1`
- `productStructureCodeTransComment.cmmt` = `Comment Line 2`
- `productStructureCodeTransComment.cmmt` = `Comment Line 3`
- `productStructureCodeTransComment.cmmt` = `Comment Line 4`
- `productStructureCodeTransComment.cmmt` = “” (blank)
- `productStructureCodeTransComment.cmmt` = `Comment Line 6`

20 Process the QDoc message that you have created in the new soapUI project and verify that the Product Structure Code is updated successfully.

Chapter 6

Business Objects and Profiles

Business Objects



A business object is a set of related data, such as the data that makes up a sales order or a customer record. In the source application, the data that makes up a business object usually resides in a set of related tables, which come from the database(s) defined in the source application. This set of tables may be maintained in a single menu program. In QAD Enterprise Applications and most other source applications, menu programs are the closest representation of a business object.

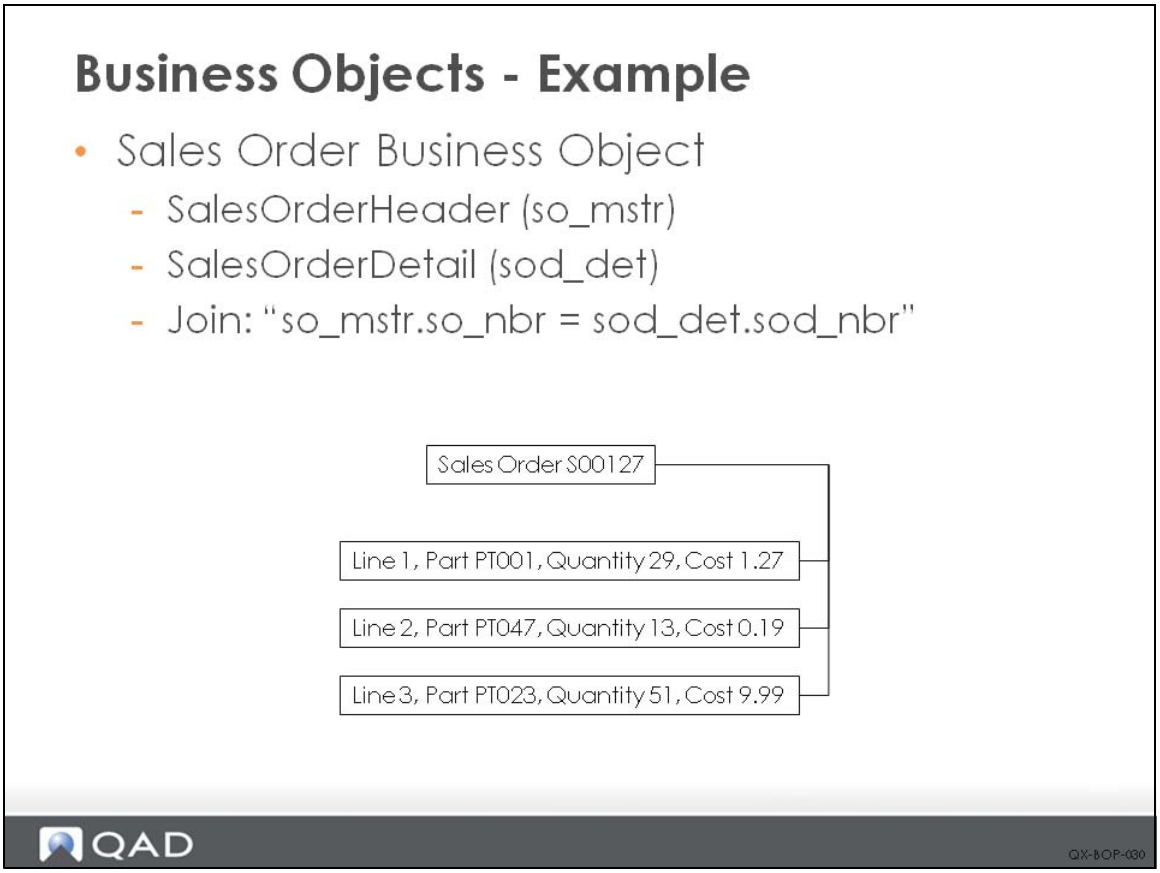
The term *business object* is used to represent both the definition of a business object, and an instance of that definition.

Business objects are specific to application type. They are uniquely identified by a combination of application type and business object name. This is because they depend on database metaschema.

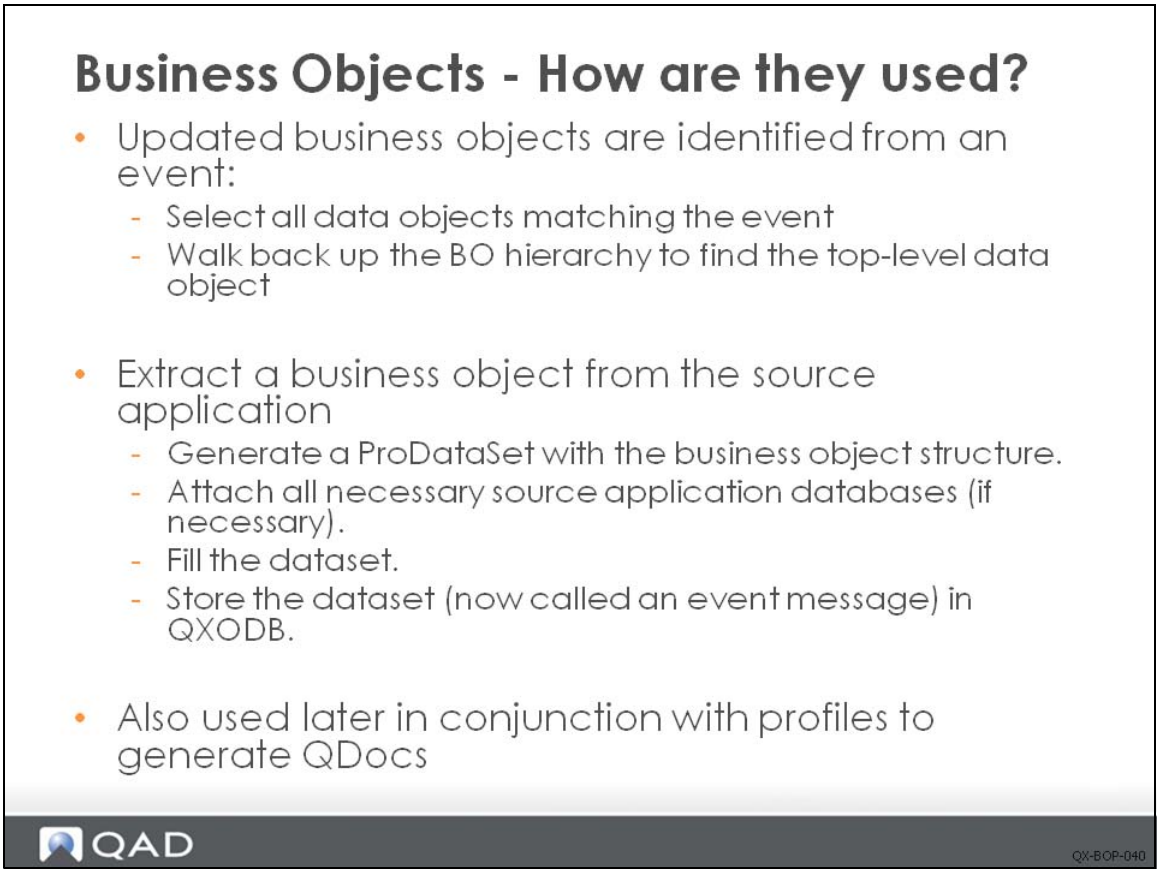
Business objects can be mapped to a Progress ProDataSet internally, and are defined using the QXO UI.

Because not all subscribers need an entire business object, you can tailor which components of a business object to include for a subscriber by defining a profile. Profiles are described later in this chapter.

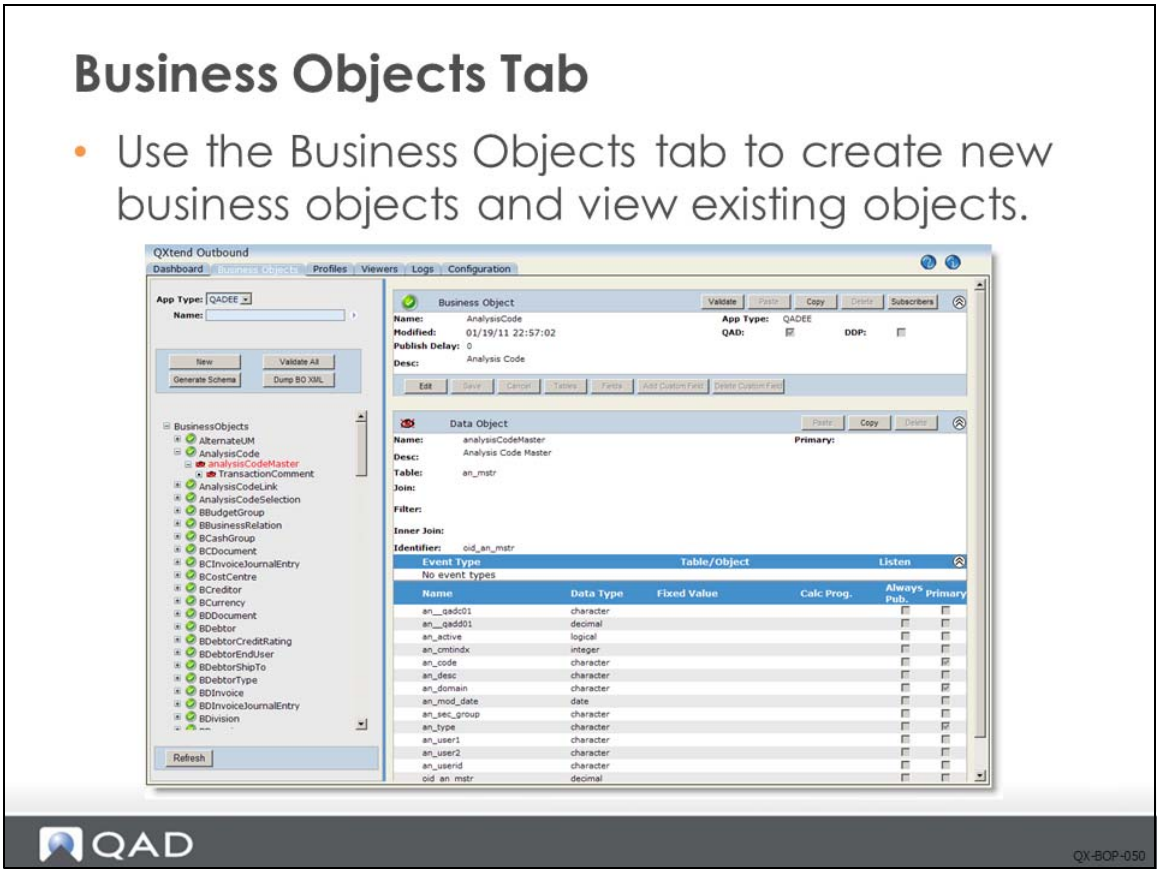
Business Objects Example



Using Business Objects



Creating a Business Object



When you click the Business Objects tab, the Business Objects screen displays. You use the Business Objects tab to create new business objects and view existing objects.

Business Objects - Building

- Select the Application Type and click New

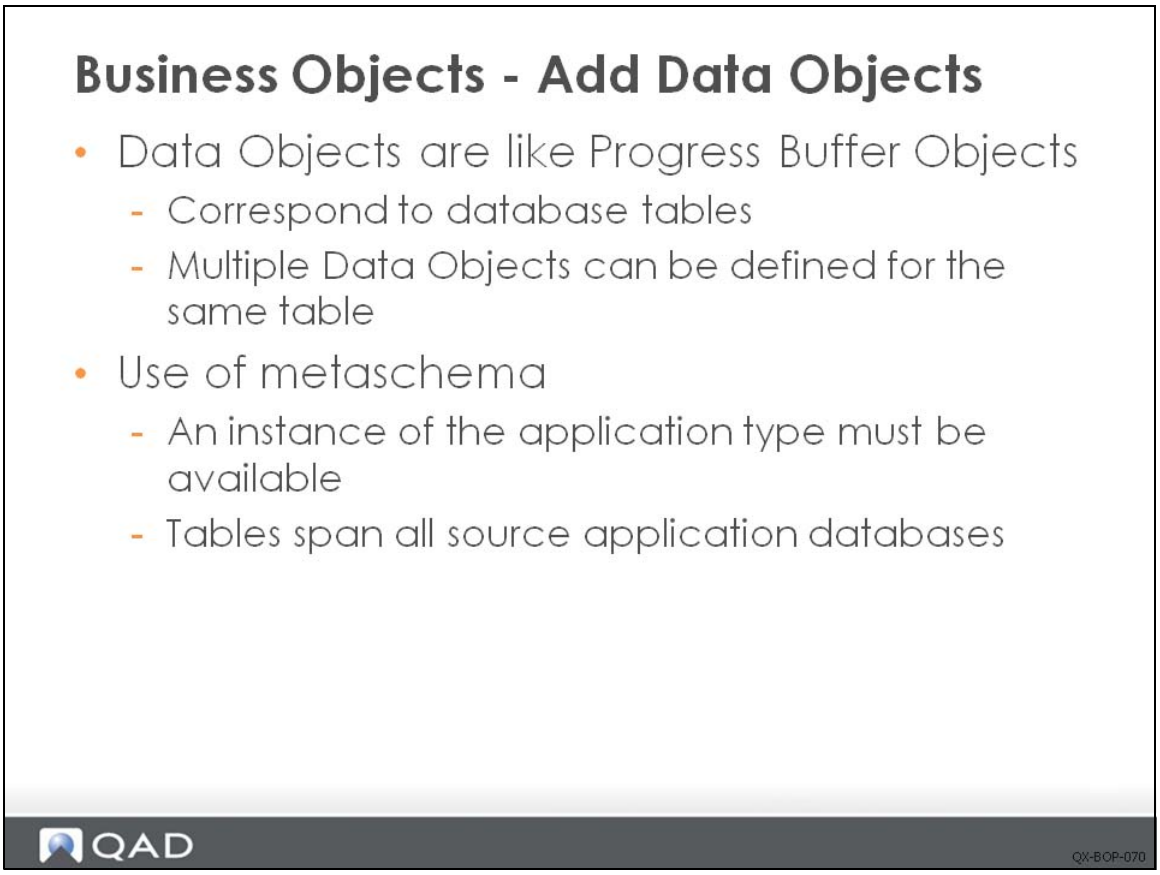


- Editable fields
  - Name (unique to application type)
  - Publish Delay (explained later)
  - Description
- Click the Tables button to add data objects to the business object.
- Colors indicate the validation state of BO
  - Green: Validated and OK
  - Yellow: Validated with warnings
  - Red: Not yet validated or has errors

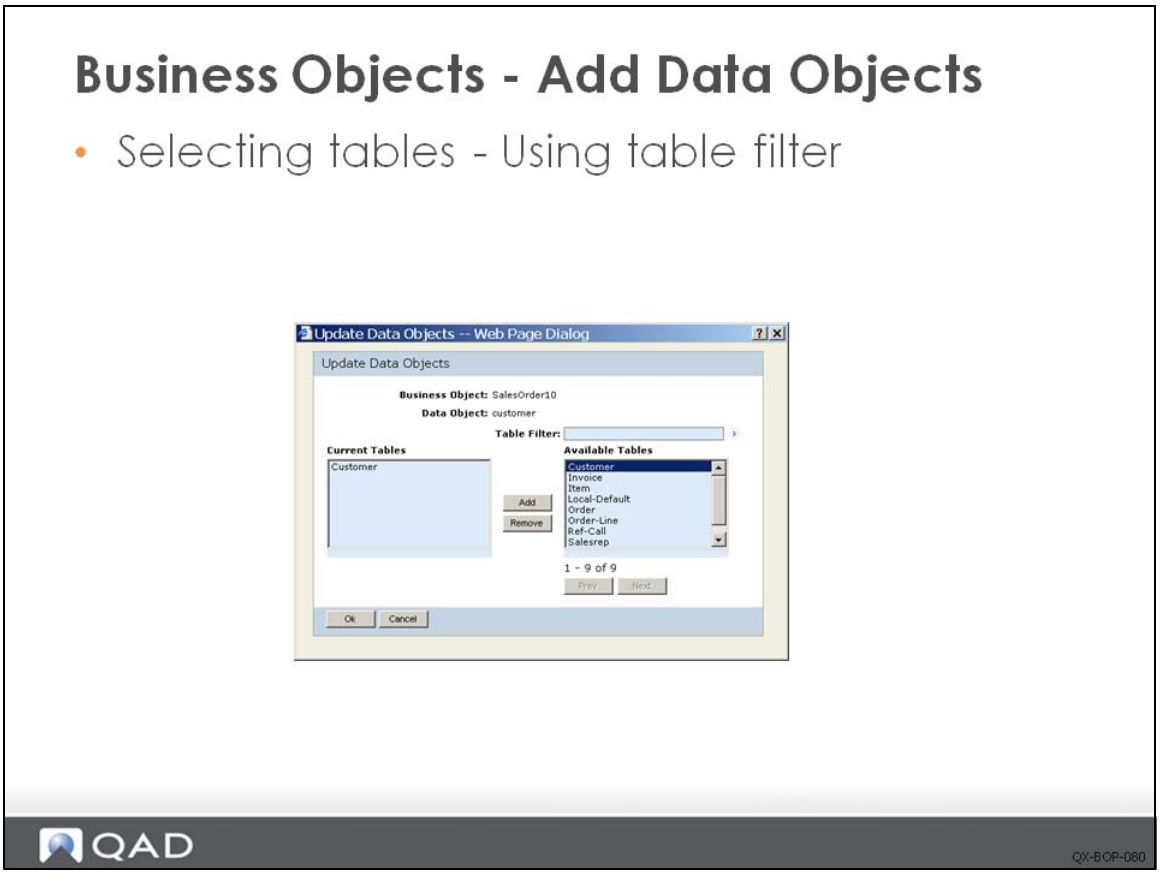


QAD

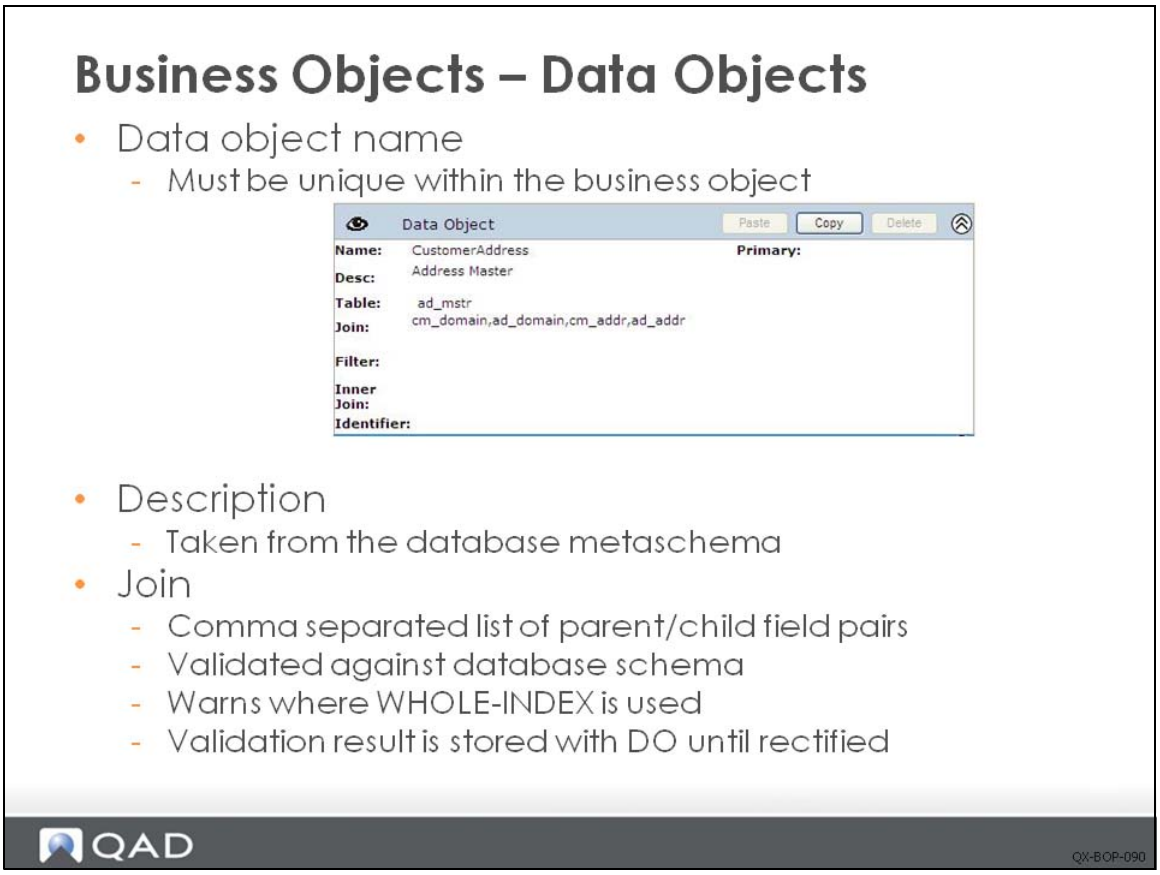
QX-BOP-060



Once you have created or modified a business object, you can modify the related data objects. Data objects are like Progress buffer objects. Data objects correspond to database tables. You can define multiple data objects for the same table.

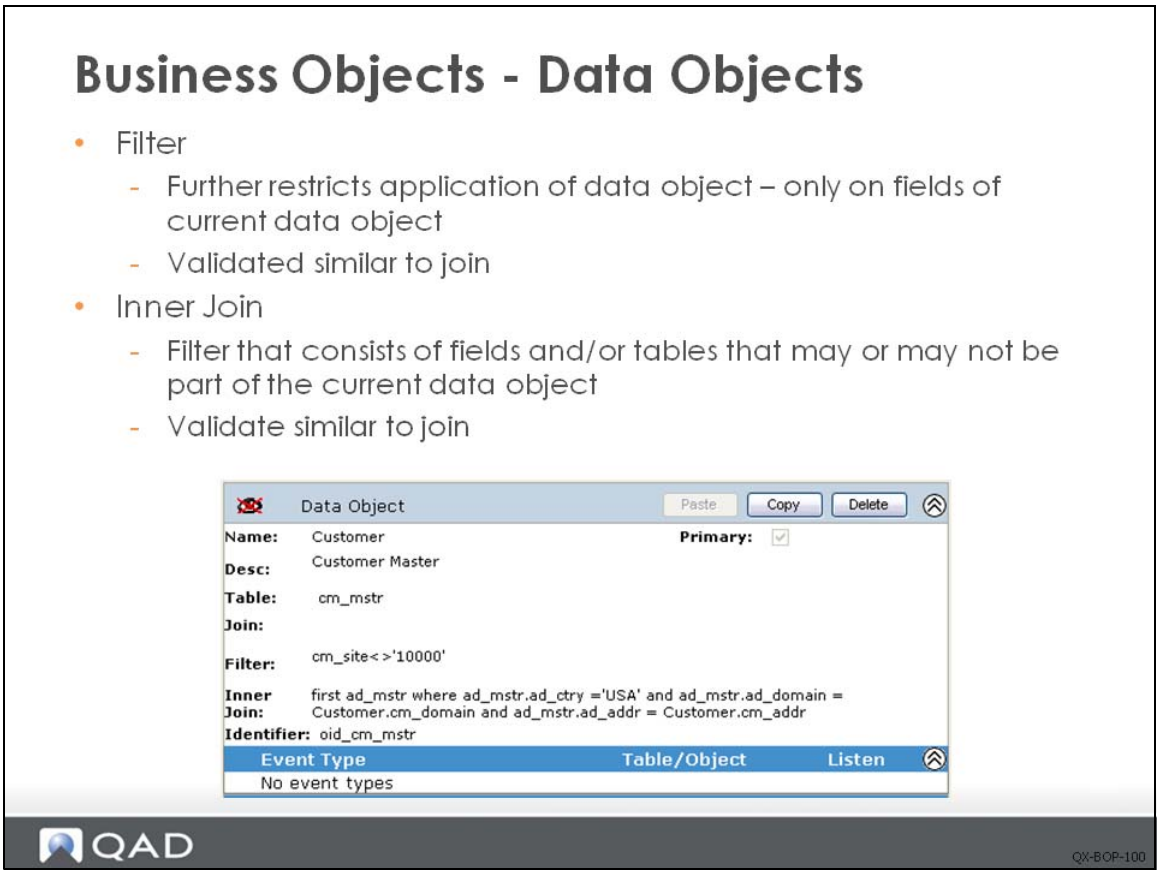


Optionally, enter filter criteria to limit the data stored after extraction from the application. The filter uses Progress 4GL expressions.



The data object name must be unique within the business object. This allows you to have the same table defined in the business object more than once but with a different data object name. The data object name is used as the buffer name when building the business object during processing.

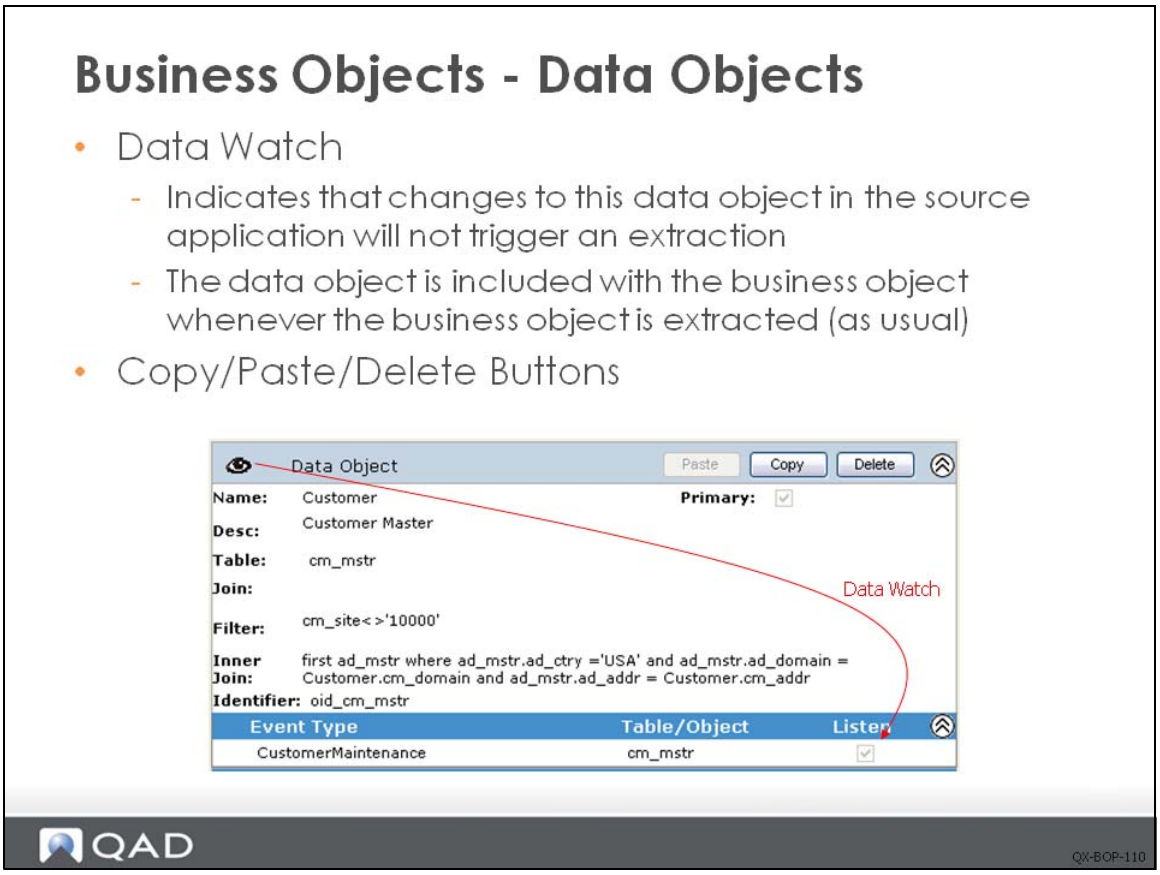
If the available indexes in one of the source application tables does not sort the records correctly for the searches being performed, whole-index search may occur. In these instances, the entire index may be scanned. The search still works but takes much longer to complete. In the example of CustomerAddress, if Join is defined as cm_addr, ad_addr, then a warning will be given during validation as whole-index search may occur when extracting the data object. This is mainly for performance consideration.



Optionally, enter filter criteria to limit the data stored after extraction from the application. The filter uses Progress 4GL expressions.

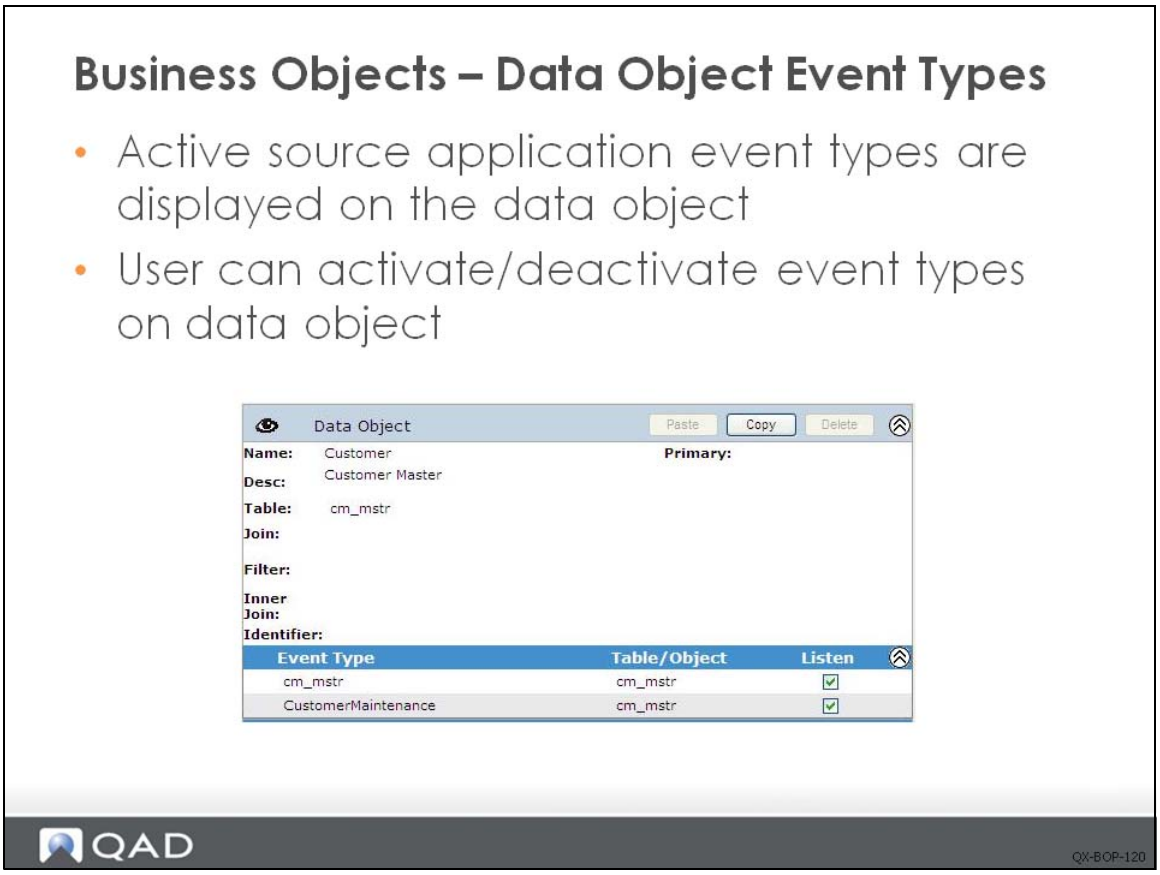
An inner join is essentially a filter that consists of fields and/or tables that may or may not be part of the current data object. Using inner joins enhances your ability to extract the types of data you want from business objects in your system.

In the example, you create the inner join query to extract customers only in USA. You use inner joins to query fields in other business objects in your system - address information, not related to the current customer data object.



To streamline the creation of business objects, you can copy and paste data objects from one business object to another. For example, if you are creating a new object that references an address, you can copy the address data object associated with a different business object and paste it into the one you are creating.

Note You cannot copy and paste data objects from business objects that employ direct data publishing.

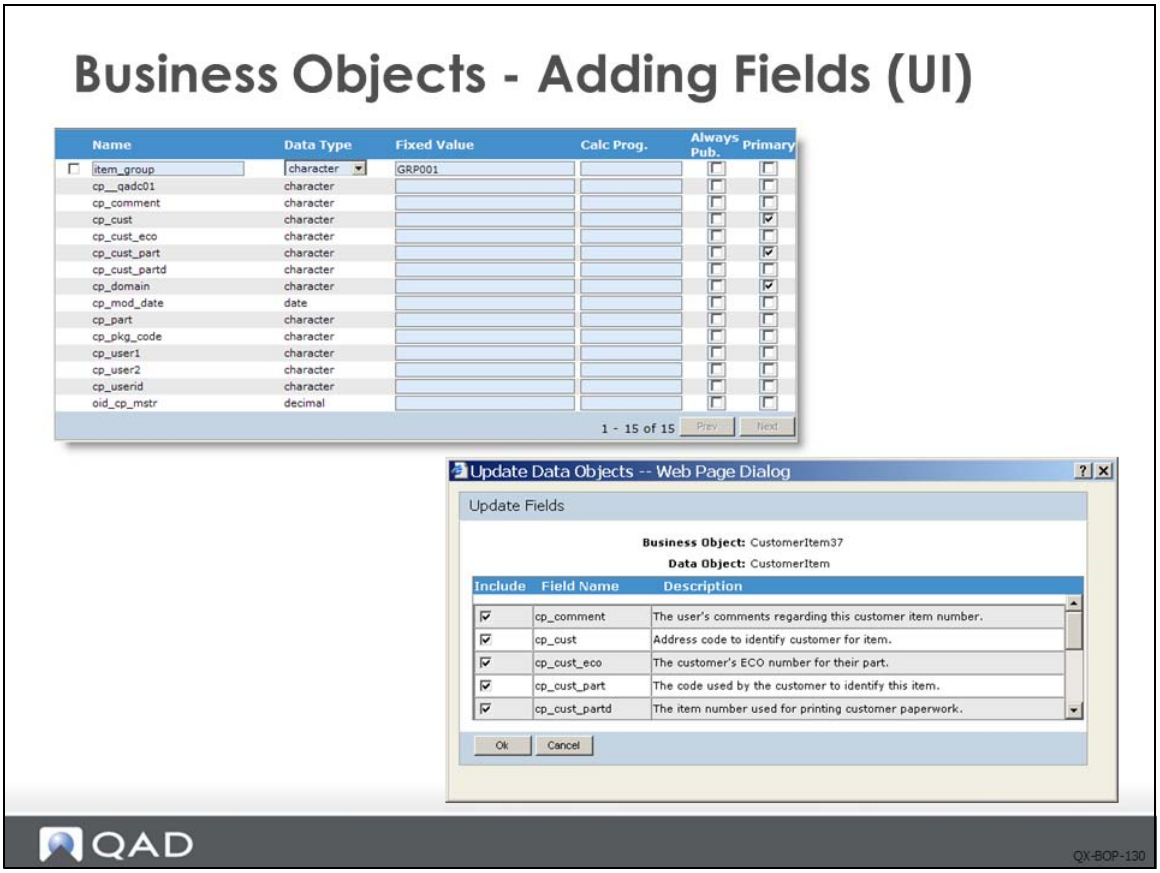


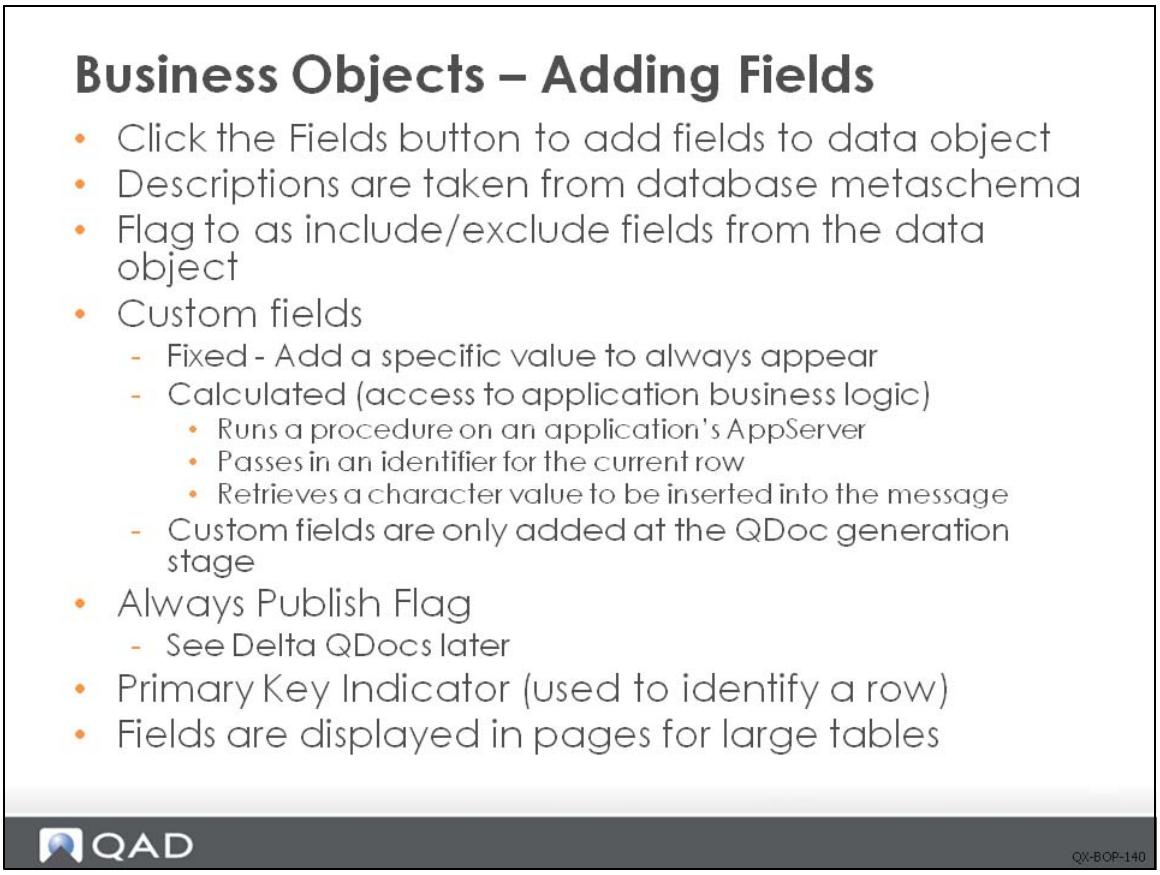
You can now view and configure event types – for example, database triggers and business events – for data objects on business objects; only active events are displayed.

User can decide whether the data object listens to a particular event type.

For direct data publish business objects, all valid event types display for the current business object in the primary top-level data object.

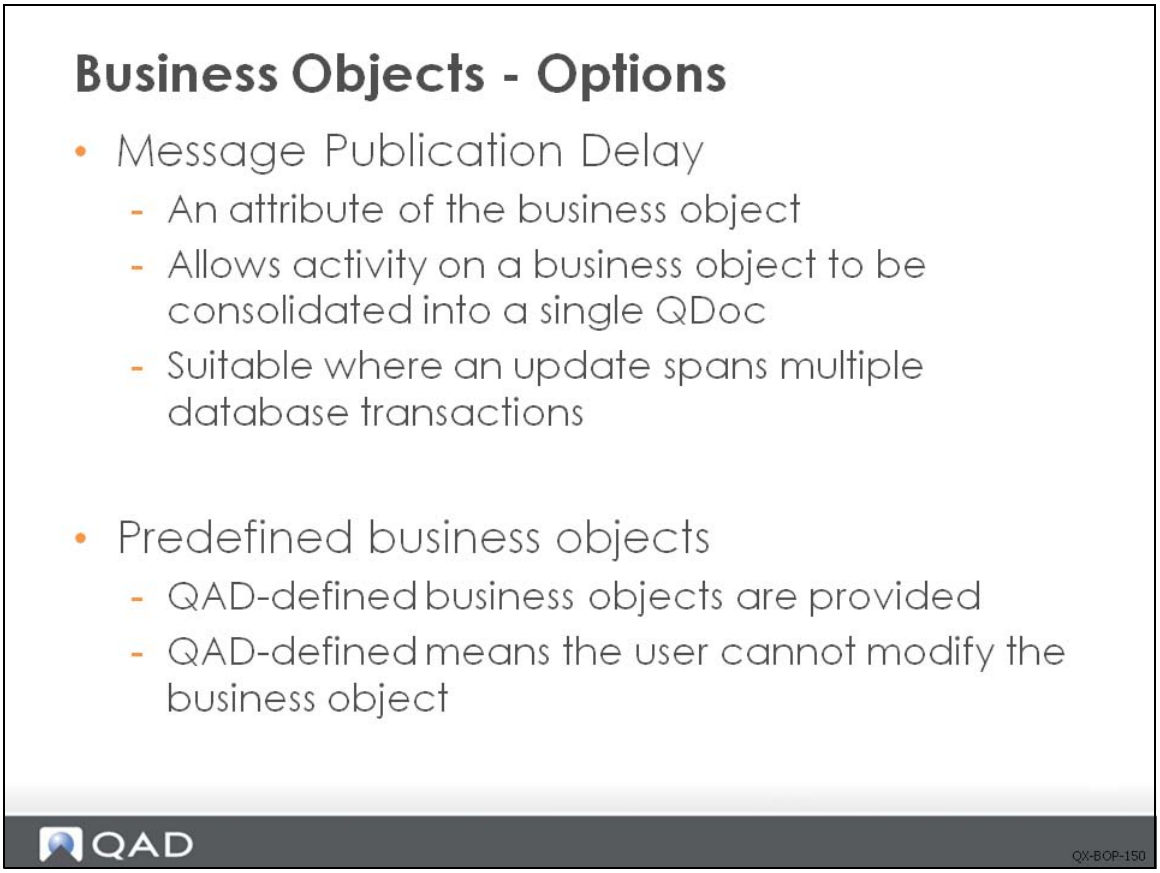
Adding Fields





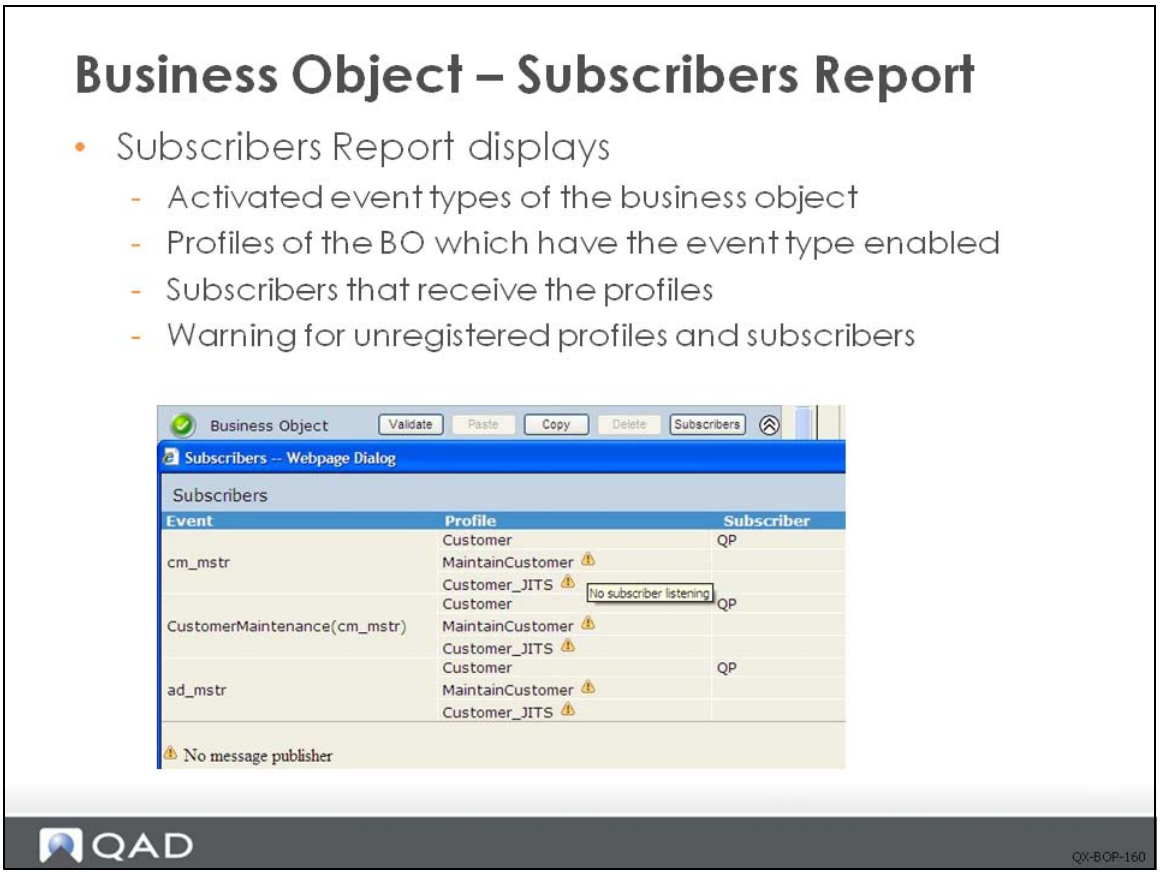
There is a performance consideration when designing business objects. Only the fields that are required for processing should be included in the business object definition. This helps limit the amount of data extracted from the source application and stored in the QXO database.

Options



Set the message publication delay setting to a nonzero value for business objects that may generate multiple notifications to the `qxevents` database prior to completion of data entry. For example, sales order entry creates one or more notifications as different tables are updated before data entry is complete. Setting the value to 60 seconds or more allows time for data entry to complete and only the final data to be exported.

QAD-defined business objects cannot be modified. If you want to customize these object, first copy the object, rename it, and then save the definition. You can then select the copied object and click Edit to modify the object as required.



You can now display a Subscribers report to determine the subscribers that will receive messages based on a business object and the events that will trigger messages. The report displays the following:

- Activated event types
- Profiles related to the current business object that have this event type enabled
- Subscribers that receive these profiles based on the event type

If a profile is not assigned to a message publisher, a warning symbol displays next to the profile name. Similarly, if a subscriber is not assigned to a message sender, a warning symbol displays next to the subscriber name. A tool tip for the warning symbols provides the exact reason. A tooltip for the warning symbols indicates the exact reason.

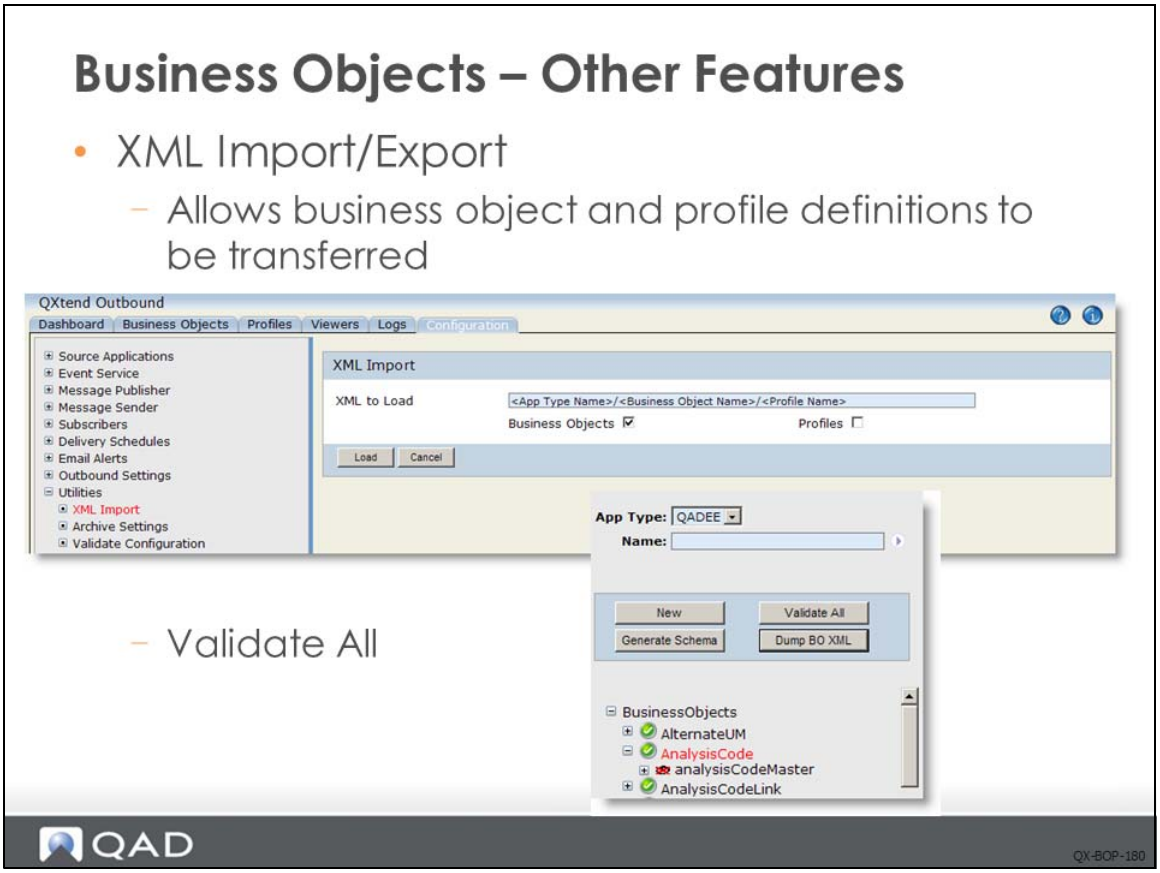
To display the report, click the Subscribers button on the Business Objects screen.

Predefined Business Objects



QAD QXtend comes with many predefined business objects.

XML Import/Export

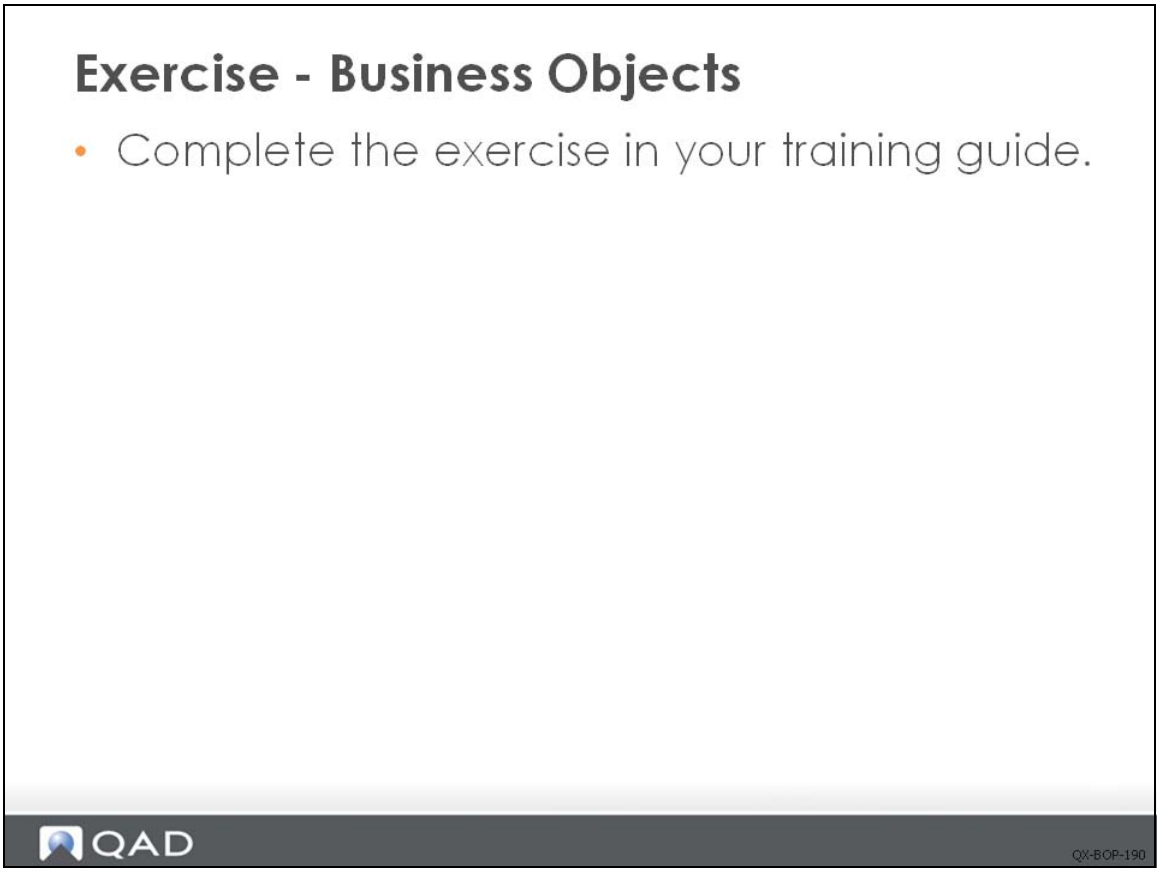


Using the XML Import feature under the Utilities node in the Configuration tab, you can import XML documents. This is required during initial system implementation to populate the database with QAD-supplied default data, and can be used to transfer XML business object and profile definitions from one QXO instance to another.

You can use the Dump BO XML button on the Business Object screen to export the definition to an XML file along with any profiles associated with the business object. These XML files can be loaded into another instance of QXO using the XML Import utility on the Configuration tab (Configuration|Utilities|XML Import).

You can use the Validate All button to validate all business objects under the selected source application type. If there are many business objects, the validation process may take a few minutes.

Exercise: Business Objects



The following list shows a number of key concepts used in the business objects in QXO. In each statement below, fill in the correct term from the list.

identifier	profile
data watch	inner join
join	filter
tables	related data
red	

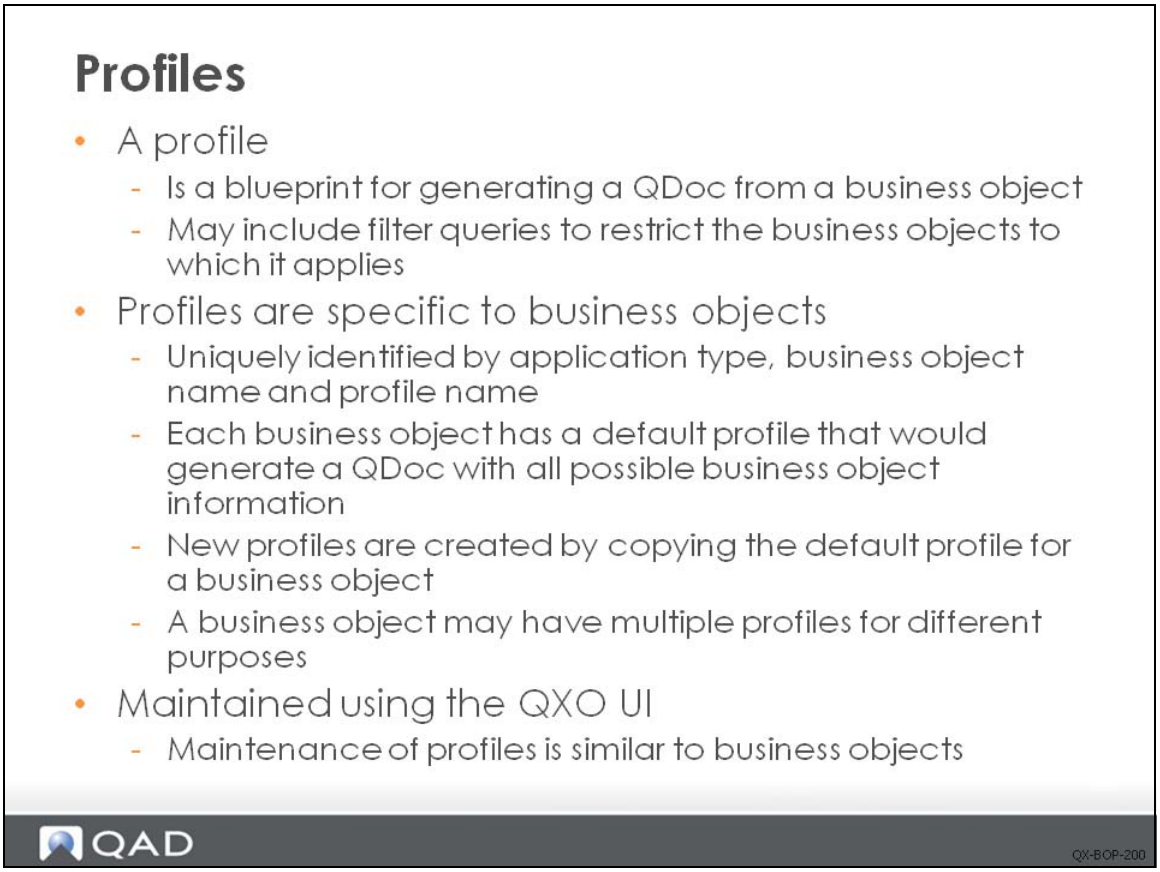
1. A business object is a set of _____, such as the data that makes up a sales order or a customer record.
2. In the source application, the data that makes up a business object usually resides in a set of related _____,
3. Because not all subscribers need an entire business object, you can tailor which components of a business object to include for a subscriber by defining a _____.
4. When you create a new business object, it must be validated. The system indicates the state of a business object through the use of colors. Unvalidated business objects are indicated with the color _____.
5. If the source application type associated with the business object does not use rowids to identify events, you must use an _____ when defining a data object.

6 _____ indicates that changes to this data object in the source application will not trigger an extraction.

7 A _____ is used to link a data object to its parent, while _____ is a _____ that consists of fields and/or tables that may or may not be part of the current data object.

8 Take SalesOrder as an example, if you want to extract SalesOrder data whose customer is on credit hold, you add _____ to the data object; if you want to extract SalesOrder data whose due date is before 12/31/2010, you add _____ to the data object.

Profiles

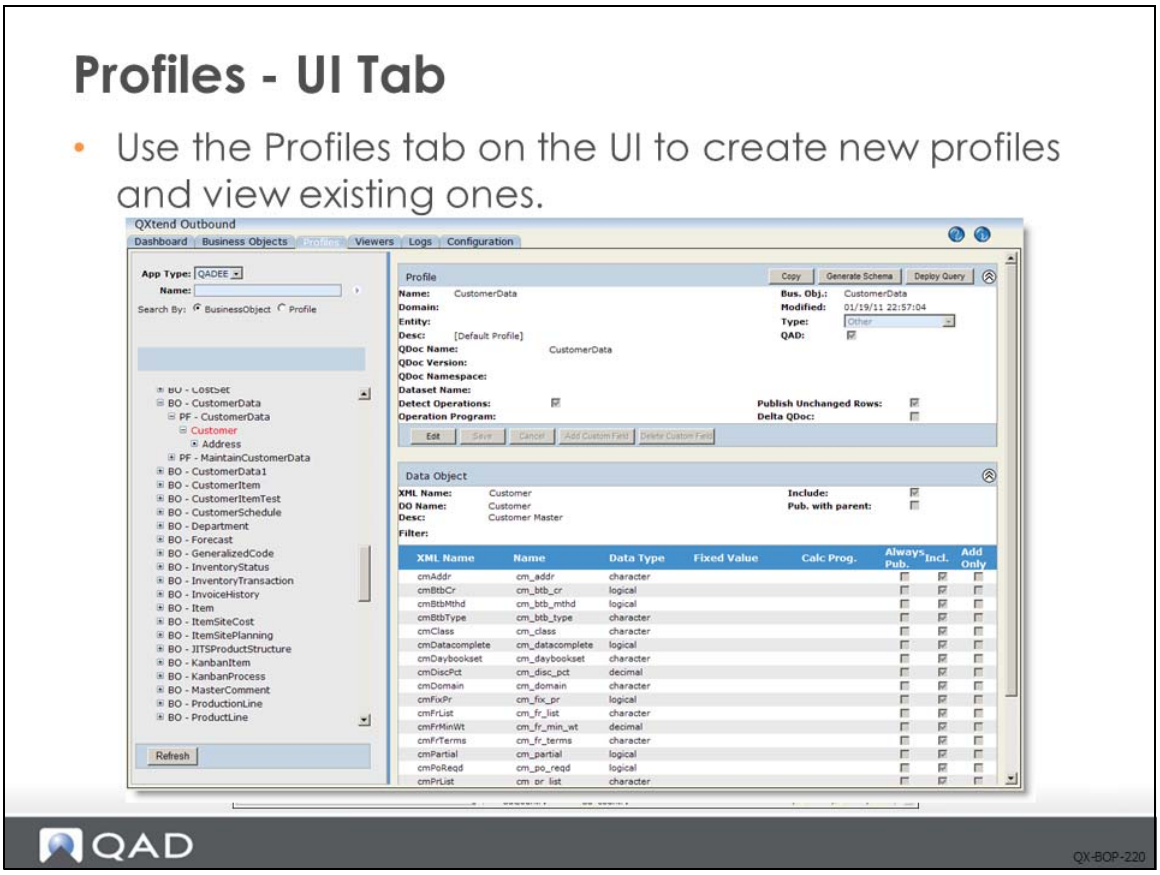


Profiles are views of business objects tailored for the requirements of specific subscribers.

To select which components of a business object are sent to which subscribers, QXO lets you define profiles. The way you define a profile is nearly identical to the way you define a business object.

Business objects and profiles are closely linked. When you create or copy a business object, a default profile with the same name is automatically created for it. When you add or delete tables or fields from a business object, the change is automatically reflected in the associated profiles as well. If you change the name of a business object, a new default profile with the same name as the new business object name is created for it. Any existing profiles associated with the business object remain unchanged.

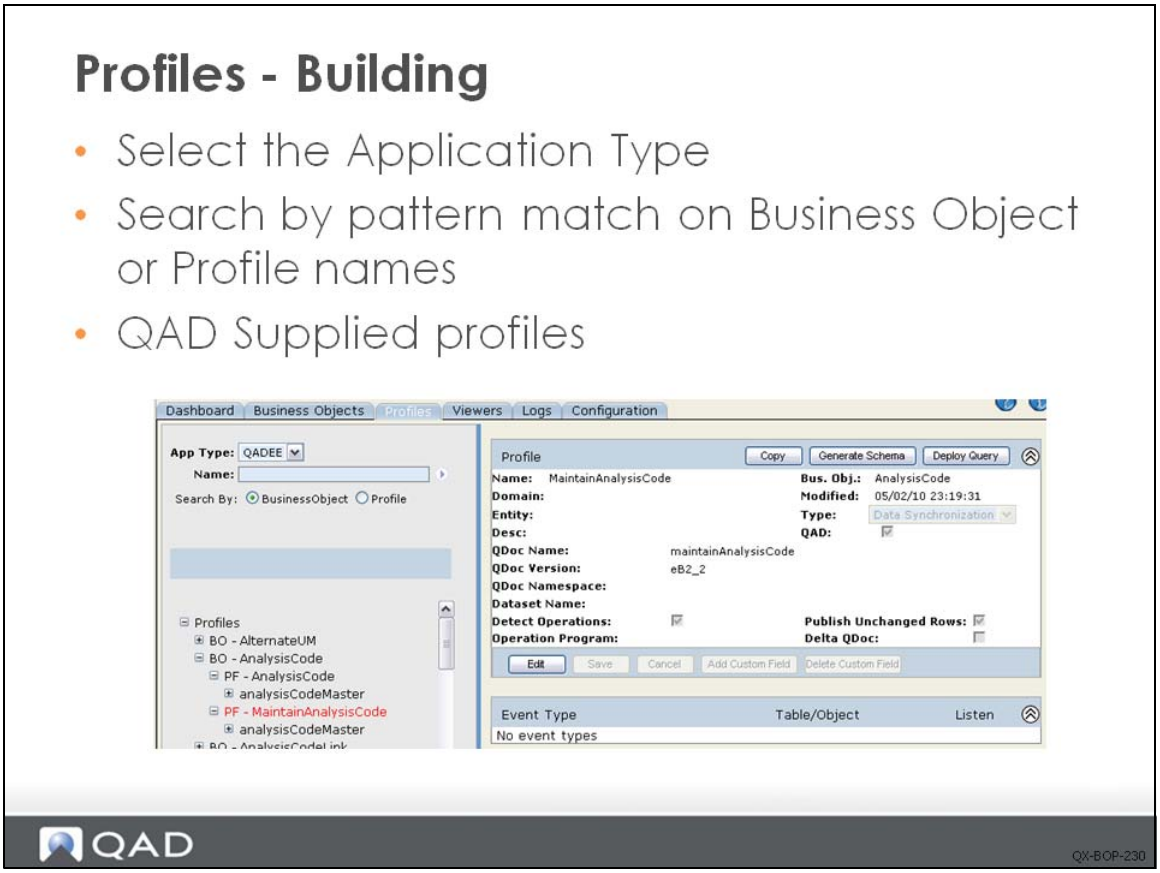

Profiles - How are they used?
• Used by Message Publisher
- Picks up an extracted Business Object (Event Message)
- Iterates through associated Profiles for the Business Object
- Generates a QDoc for each Profile & stores it back in QXODB
QAD
QX-BOP-210



Profiles are the means you use to select which components of a business object are sent to subscribers. The method of defining a profile is nearly identical to the method of defining a business object; however, unlike business objects, profiles always start as a copy of a default profile.

When you click the Profiles tab, the Profiles screen displays.

Creating Profiles



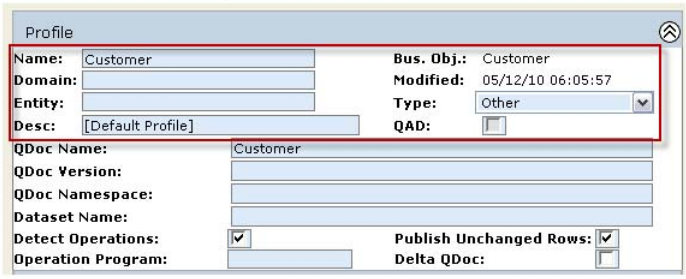
Since you cannot edit the QAD-supplied profiles or create a new one, you must copy an existing profile and modify it. Select the application type, and then search by pattern match on business object or profile name.

For a QAD-defined business object, usually you can see following QAD-supplied profiles.

- Default profile: Every business object has a default profile which has the same name with the BO. If the BO is defined by QAD, then the default profile is also supplied by QAD and it cannot be deleted.
- Data synchronization profile: This profile is used to synchronize data between different QAD enterprise application instances (or different domains). The profile's Qdoc Name and Qdoc Data match corresponding API in QXI.

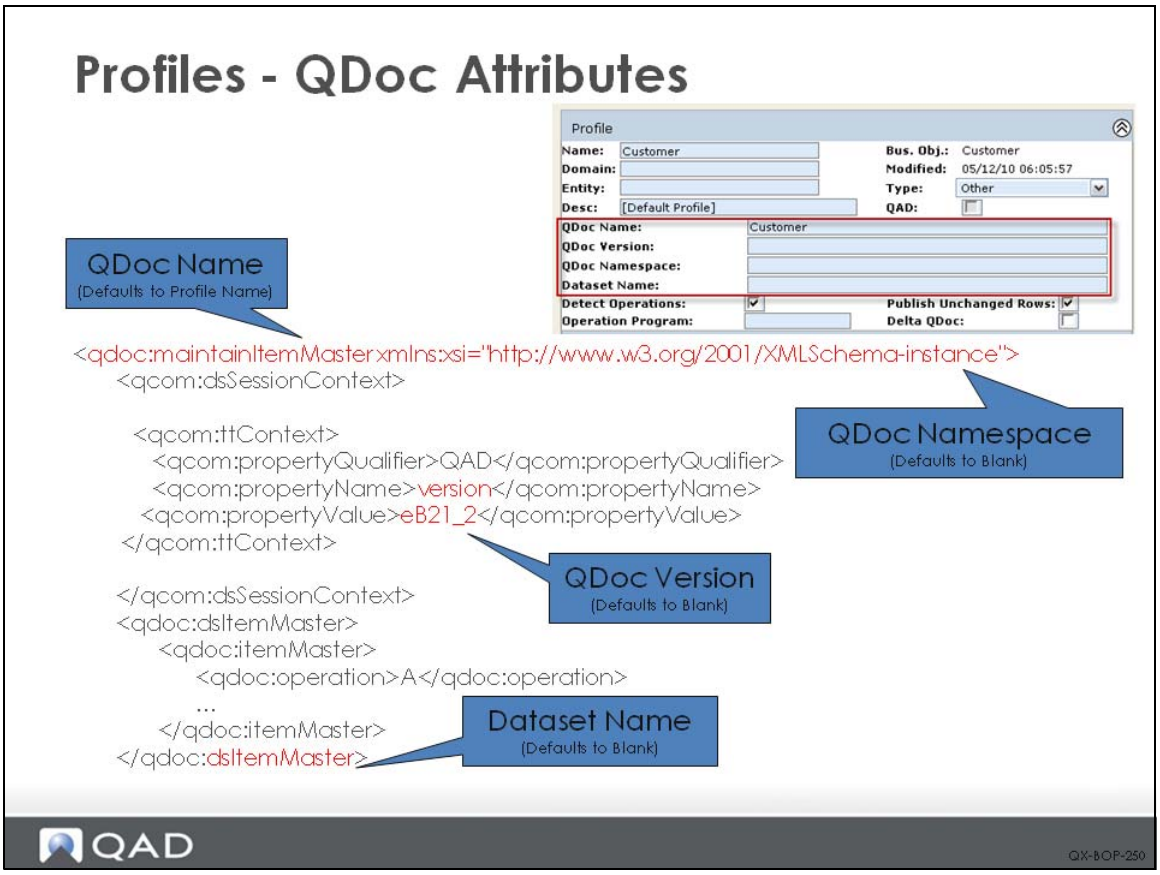
Profile Attributes

- Description
  - User defined, defaults to blank
- Domain & Entity
  - Restricts Profile to apply only to BO's from a particular domain/entity
  - The domain concept has parallels in other applications
  - In QXO, domain is set by the event trigger & could be anything
- Type: identified the response type so the response can be parsed correctly
- QAD-defined flag
  - Same as for Business Objects



QAD QX-BOP-240

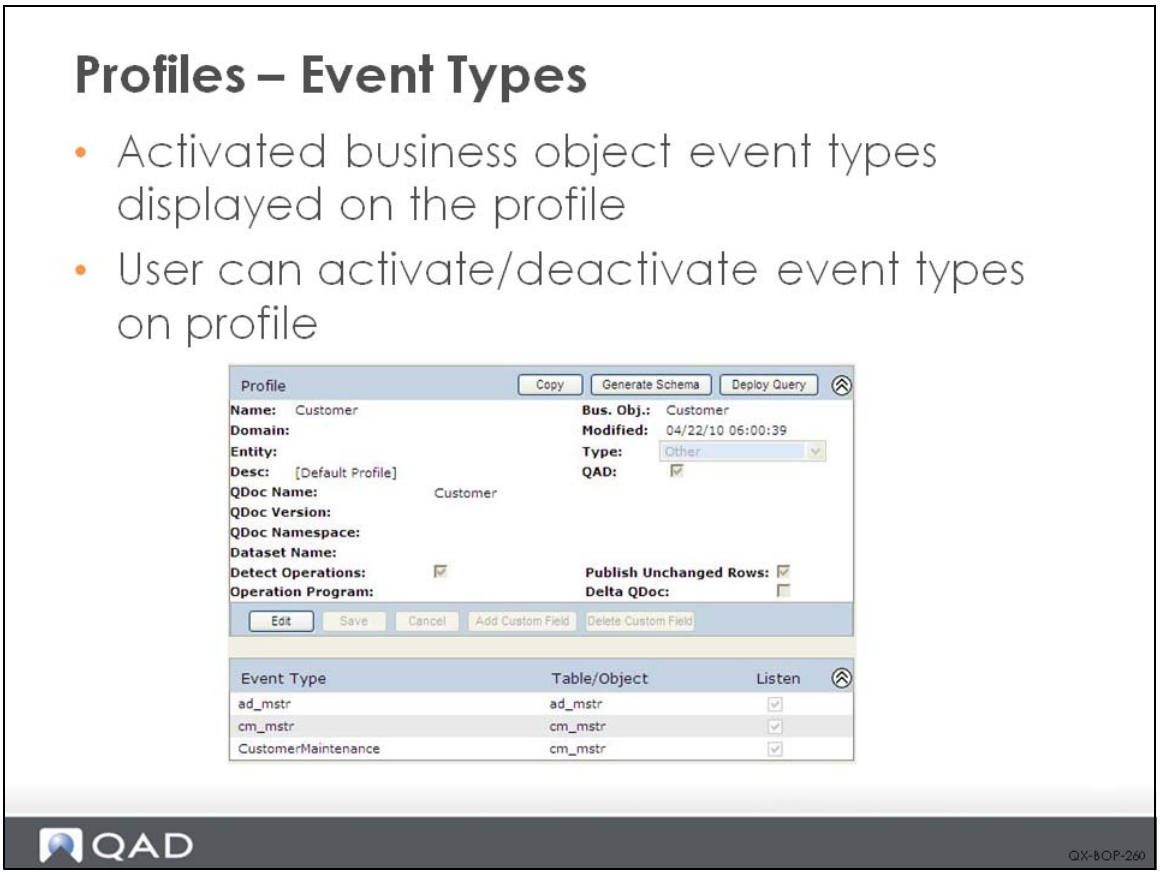
Define the required attributes, as shown above.



The QDoc name is the top level node name of the body of the request. The version of the QDoc is stored in the session context section. Look for the session context node where the `propertyName` is "`version`". The "`propertyValue`" node contains the version of the QDoc.

In the case of a QXtend Web Service subscriber, the QDoc name and version are used as part of the validation in the QXtend Inbound web service. An incoming QDoc must have a valid QDoc name and version before being passed through to the target receiver.

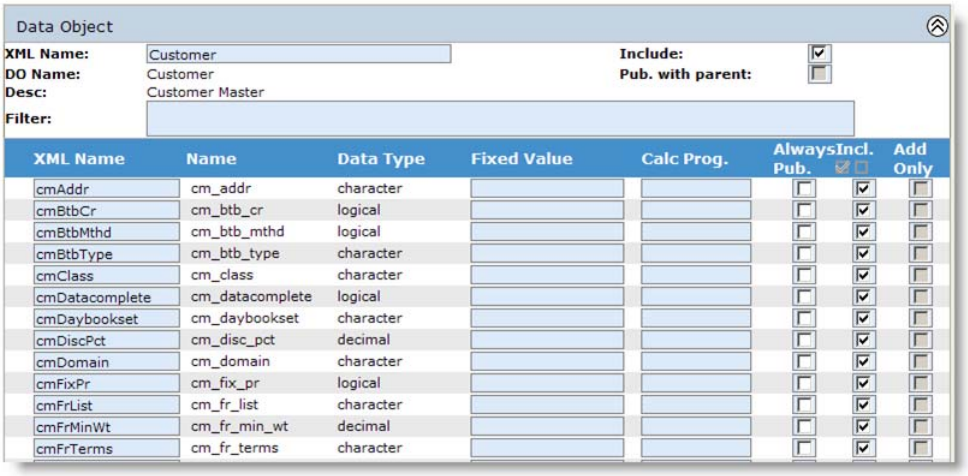
Dataset Name. Specify the name of the root node in the QDoc XML.



You can now specify which events must occur in order for a profile to be published by enabling the Listen check box for selected events. By default the Listen check box is enabled for all event types for the profile.

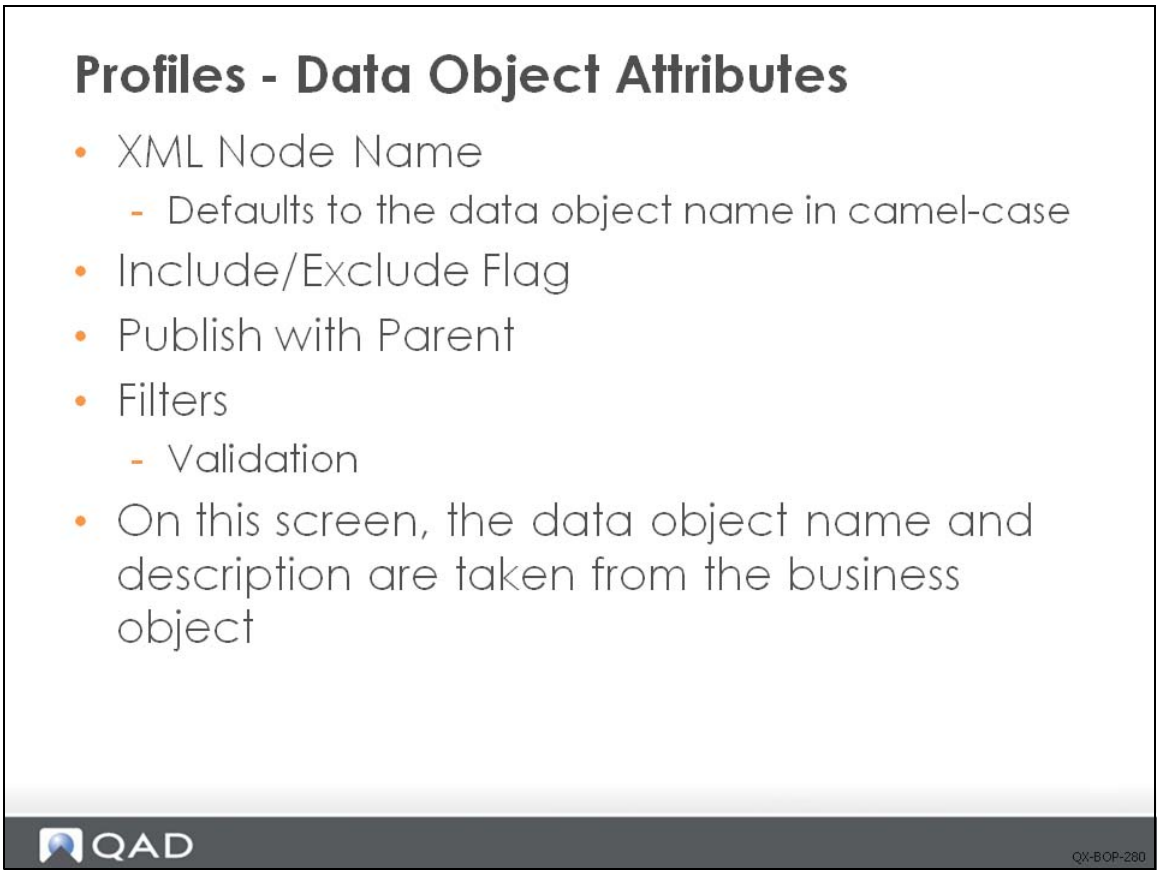
The event types that are available are the activated event types for all data objects in the business object related to the profile. Hence these event types are related to the profile as a whole, not to a specific data object. The profile event types do not appear when editing a data object for the profile.

Profile – Data Object



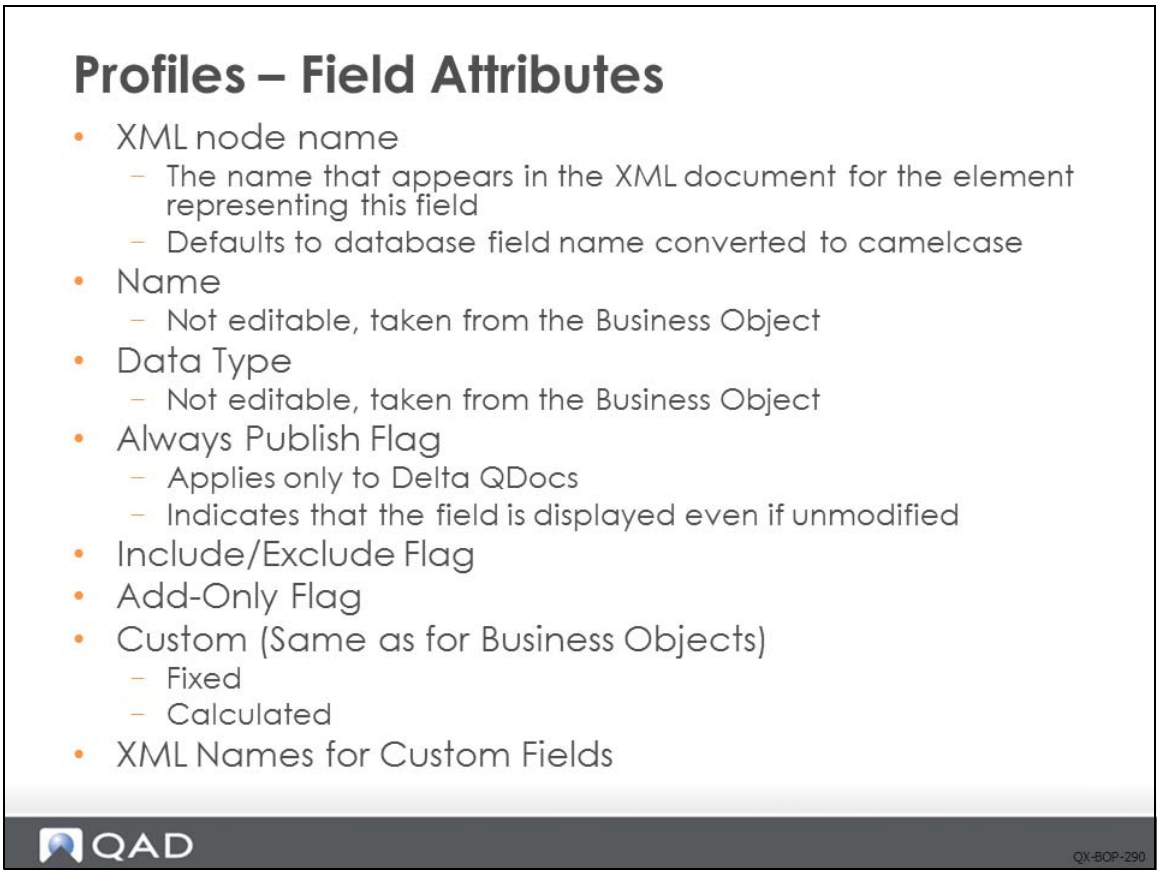
QAD

QX-BOP-270



The following data object attributes are required:

- XML Node Name: The name of the XML element or node for the data object in published QDocs. Defaults to the data object name in camel-case.
- Include: Select the check box if you want this data object to be included in QDocs generated based on this profile. This setting applies to all of the fields in the object. When you include the object, you can also determine on a field-by-field basis which fields to include.
- Publish with Parent: Indicate if you want the fields in this data object to be published automatically whenever data in the business object's parent is published. For example, when a field associated with a sold-to customer changes, you may always want to publish the sold-to customer address.
- Filter: Optionally, enter filter criteria to limit the data extracted from the application. The filter uses Progress 4GL expressions.Profiles can even filter on values that are derived from fixed values and calculated programs.



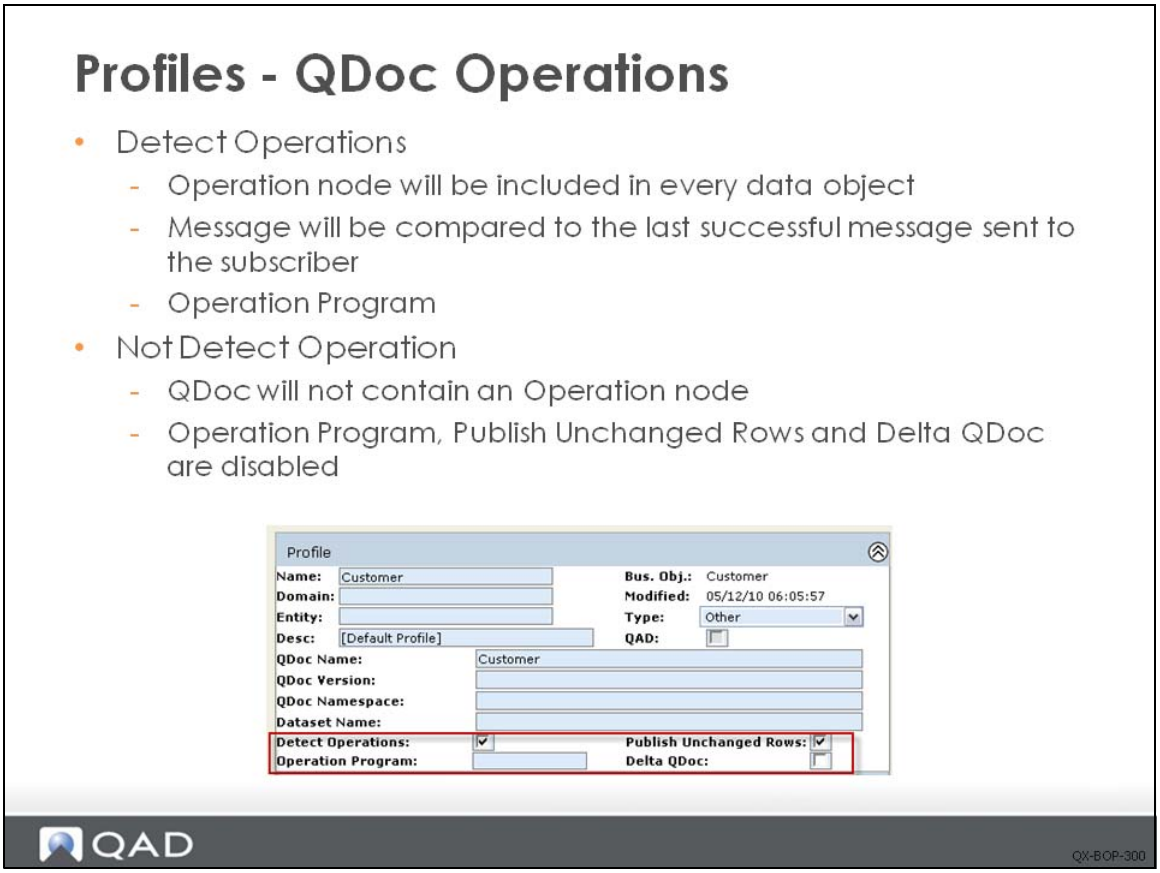
Add Only. Indicate if you want to include this field in the resulting QDoc if the operation is an ADD operation.

By using the Add Only check box, you can define the fields that are locally owned by the target application, and update transactions in the target application without overwriting locally owned data. Selecting the Add Only check box automatically selects the Include check box.

Any field that is specified as Add Only must have its Always Publish check box disabled and the Default Operations check box selected. On saving the data object, the system displays an error message for fields that have incorrect check box selections. An error also displays if Add Only is selected for a key field.

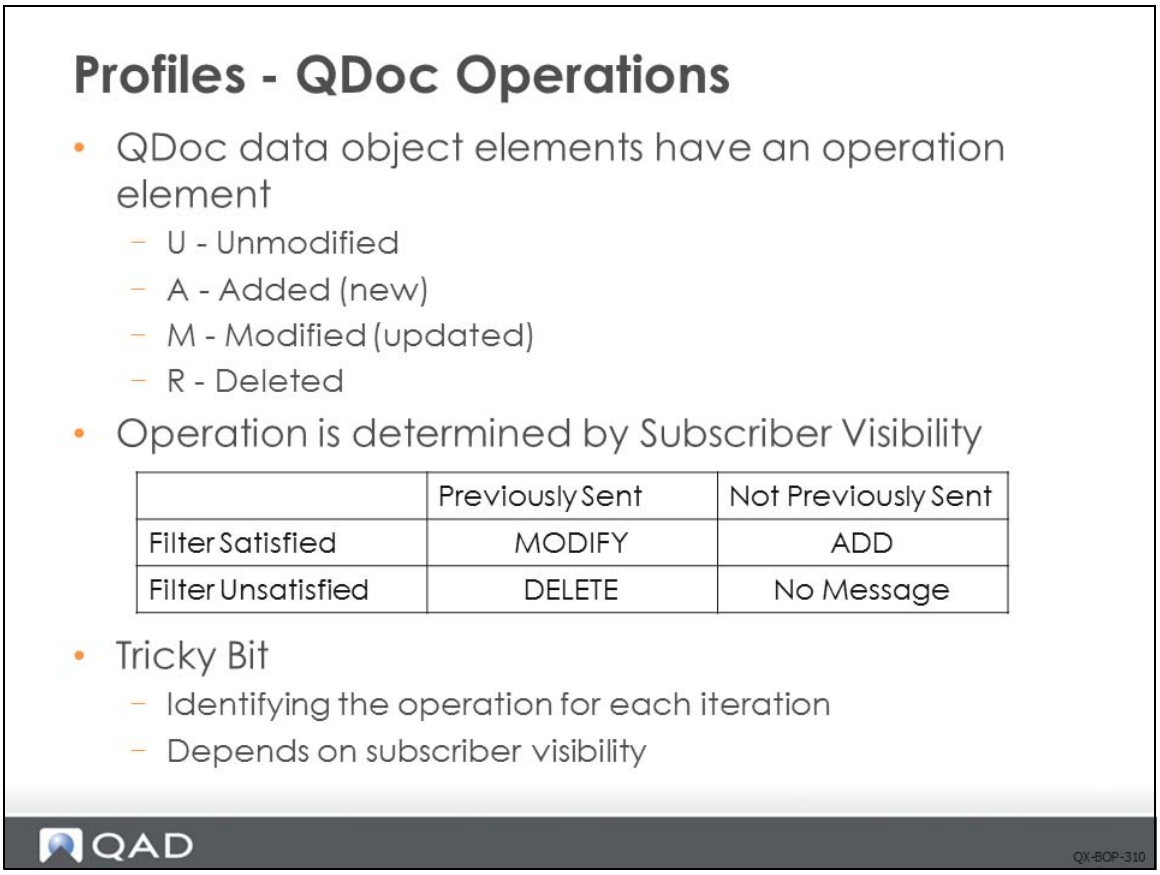
Fixed value fields can contain Progress expressions and perform built-in Progress calculations; for example, =today+2. Fields from the current buffer can be referenced in this calculation by including the XML node name surrounded by dollar ($) characters. The expression must start with an equals (=) character; for example, =substring($ptDesc1$,1,12).

QDoc Operations

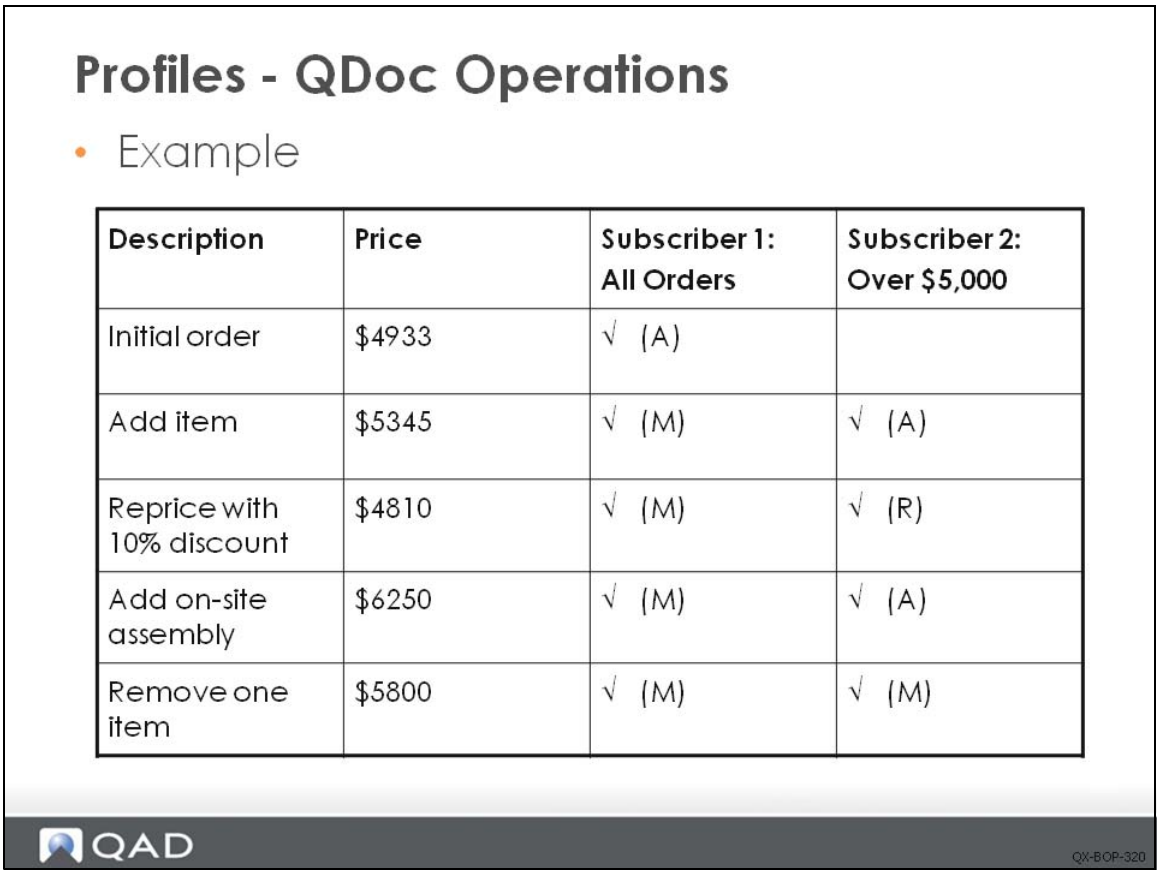


Detect Operations. If you select this field without specifying an operation program, an operation node will be included in every data object in the QDoc, and the message will be compared to the last successful message sent to the subscriber to determine the operation of each data object. If you select the field and specify an operation program, the program must determine what the operation is and enter the result into a field on each data object in the profile.

If you leave the field blank, the QDoc will not contain an Operation node and will create a subscriber message based entirely on the current raw message.



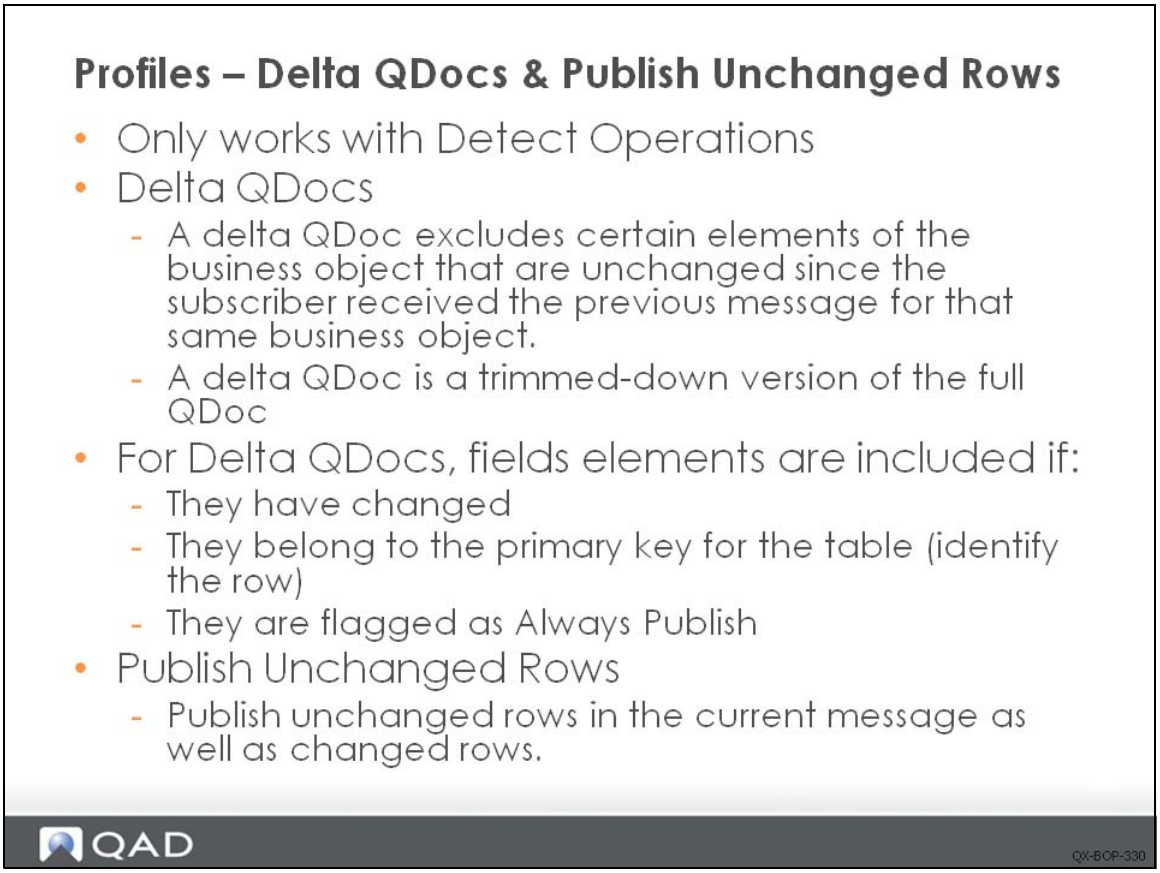
Some planning and consideration are needed when applying filters to profiles and business objects. Depending on what filters you define in the business objects or, more typically, in the profiles, a subscriber may have interrupted visibility to some data changes.



The table shows the QDocs generated for one sales order for two different profiles; one receives all sales orders and the other filters only for sales orders over $5,000.

Subscriber 1 sees all the changes, while Subscriber 2 only sees the two changes that temporarily raise the so_price value above $5,000.

Delta QDocs

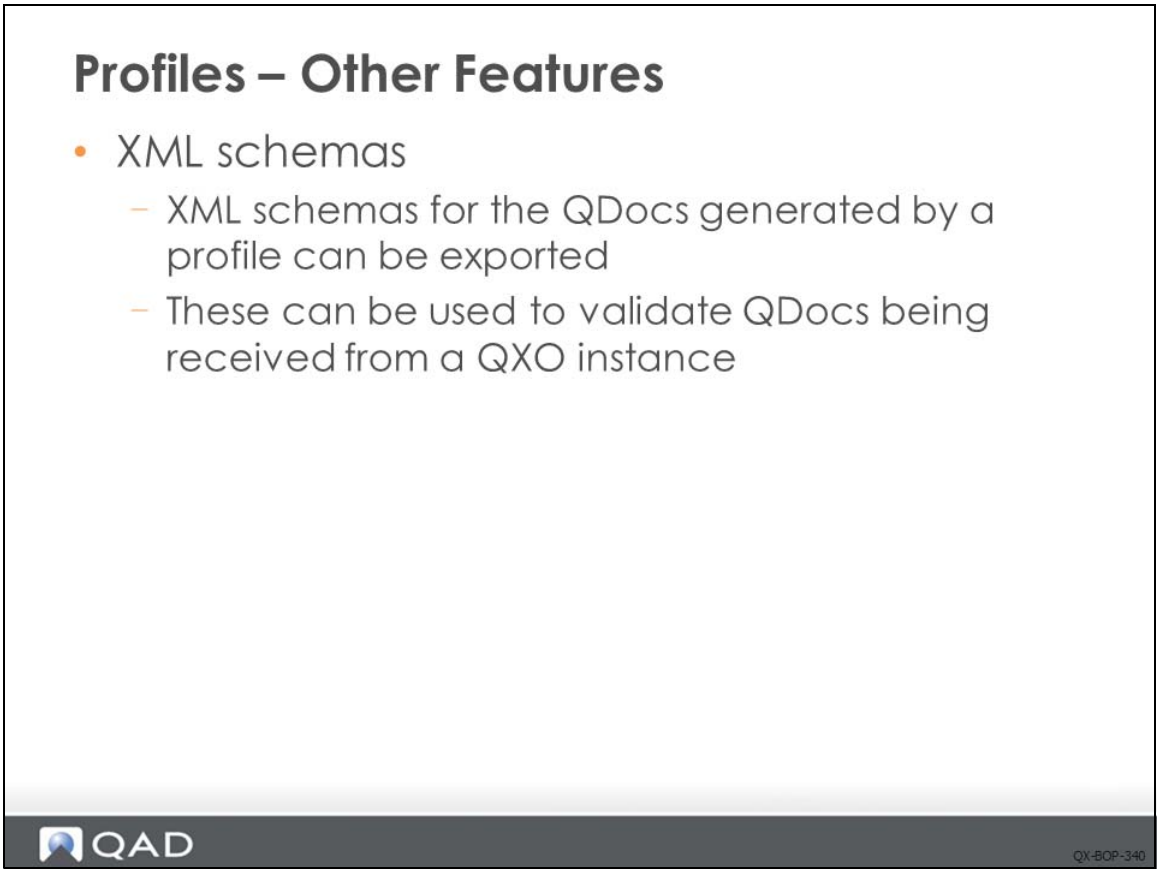


Delta Qdoc. Indicates whether this profile will send delta QDocs or complete QDocs. A delta QDoc excludes non-primary elements of the business object that have not changed since the last update. If a field is tagged Always Publish in the business object, it overrides the delta setting here.

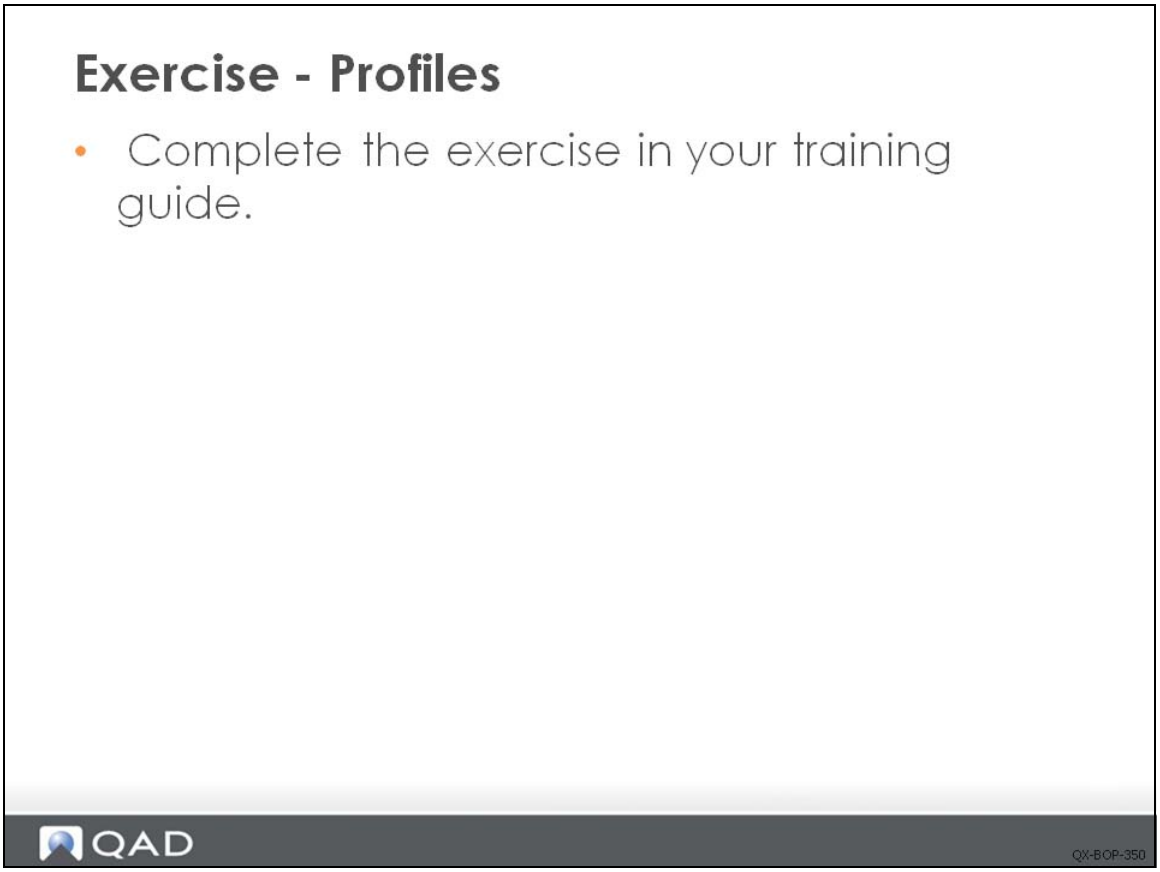
Publish Unchanged Rows. Select the field if you want to publish unchanged rows in the current message as well as changed rows. To publish changed rows only, clear the check box.

Publish Unchanged Rows only works with Detect Operations and operates in a similar way to delta QDocs, except at the table rather than the field level.

XML Schemas


Profiles – Other Features
• XML schemas
– XML schemas for the QDocs generated by a profile can be exported
– These can be used to validate QDocs being received from a QXO instance
QAD
QX-BOP-340

Exercise: Profiles



The following list shows a number of key concepts used in the profiles in QXO. In each statement below, fill in the correct term from the list.

message publisher	fields
add-only	always publish
tables	profiles
qxodb	QDoc
delta QDoc	A
M	R
publish unchanged rows	

1. You use _____ to tailor a view of a business object for a specific subscriber.
2. If you add or delete _____ and/or _____ from a business object, these changes are automatically reflected in the associated profile.
3. Profiles are used by the _____.
4. The message publisher picks up the extracted business object and iterates through the profiles associated with the business object. It then generates a _____ for each profile and stores the QDoc in the _____ database.
5. A _____ excludes non-primary elements of the business object that have not changed since the last update.

6 When Detect Operation is set to true, and no previous subscriber message of a particular Item data has been sent to target application, then, if you update this item, QXO will deliver a subscriber message with operation _____; Then, if you change this item, QXO will deliver a subscriber message with operation _____; finally, when you delete the item, QXO will deliver a subscriber message with operation _____.

7 If you want to synchronize data to target application but don not want to overwrite locally owned data in target application, you set those locally owned data fields as _____.

8 When delta Qdoc is set to true, if you want to publish a child data object even it has not been changed, you set _____ to true; if you want to publish a field even it has not been changed, you set _____ to true.

Lab: Business Objects and Profiles

QAD QXtend 1.7.2 has been installed with both QXI and QXO components. In the previous labs you performed the basic configuration that is required to enable requests to be processed by QXI and QXO. In the following lab exercises you will build your own business objects and profiles that can be used to pass data to QXI.

`<LabHomeDirectory> = C:\QXtendTraining\Labs\06-BusinessObjects\`

In the QXO Configuration lab you learned how to synchronize Generalized Code data from a master domain to several target domains using the standard business objects and profiles that were shipped with QAD QXtend. The tools provide by QXO allow you to build complex business objects that reflect the data structures in the source application. The profiles then map the business object into the format required by the subscribing applications. The QXO toolset provides a powerful, flexible mechanism for accessing data in QAD EA.

1. Generalized Code Data Synchronization

In the QXO Configuration lab exercises you have already performed the synchronization of Generalized Code data between domains in QADEE. However, in the QGen lab you created a new schema for Generalize Codes Maintenance and you need to use this new schema for the synchronization of the data. Normally you would modify the business object if necessary, create a new profile, and change the necessary details including the version number. For this lab you will create a new business object and profile, and then configure QXO to synchronize the data using your new business object and profile.

The Generalized Code API is a good example to start with.

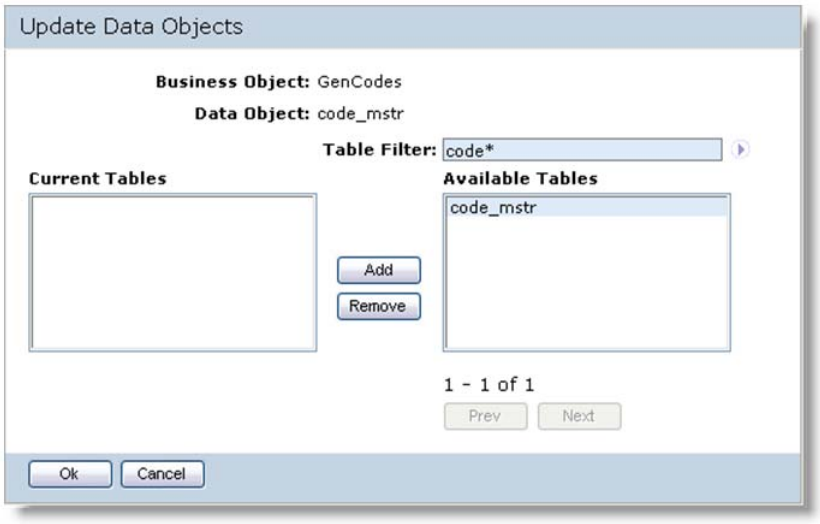
1.1 Create New Business Object

First create a new business object that specifies the data that needs to be extracted in order to create a QXI QDoc that can be processed successfully:

1. Open the QXO Web application in Internet Explorer: `http://qaddemo:8080/qxo`.
2. Select the Business Objects tab.
3. Ensure that the App Type is set to QADEE.
4. Click the New button.
5. Click the No button for “Create a direct data publish business object?”
6. Enter the following details for the business object:
   a. Name = `GenCodes`
   b. Description = `Generalized Codes business object created for QAD QXtend training`
7. Click the Save button.

Add Tables to the Business Object

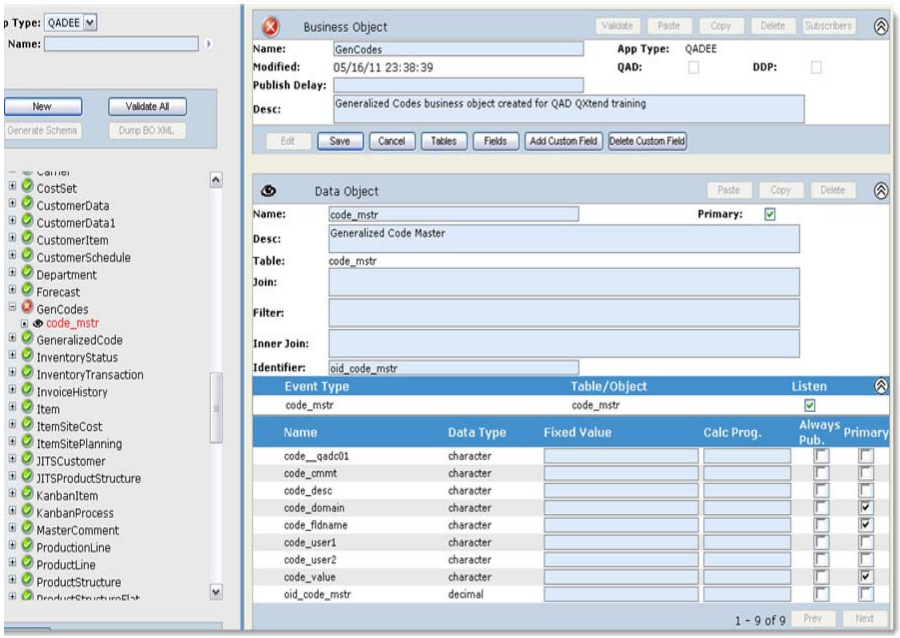
1. Select the GenCodes business object from the Business Objects menu.
2. Click the Edit button.
3. Click the Tables button.
4. Enter `code*` into the Table Filter and click the Fetch button.



5. Select the `code_mstr` table and click the Add button.
6. Click the OK button.
7. Click the Save button.

Edit the Fields

1. Select the GenCodes business object from the Business Objects menu.
2. Drill down to the `code_mstr` data object.



3 Click the Edit button.

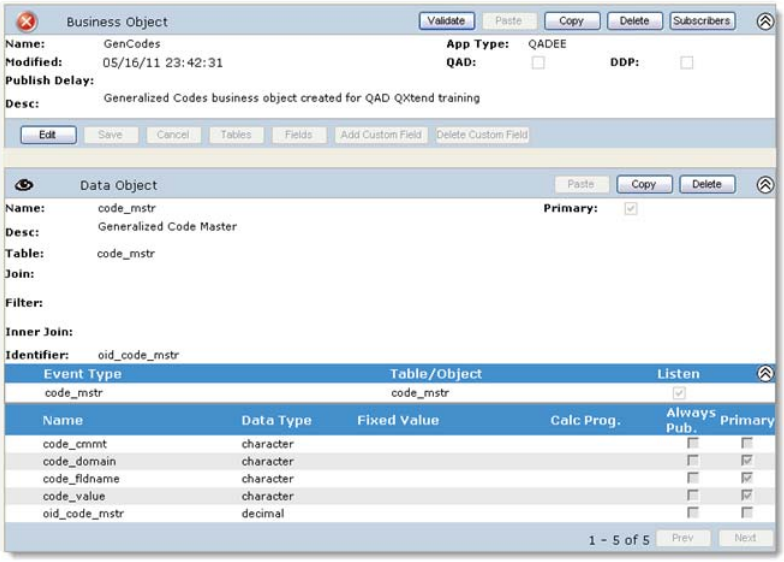
4 Click the Fields button.

5 Ensure that only the following fields are included:

- `code_cmmt`
- `code_domain`
- `code_fldname`
- `code_value`
- `oid_code_mstr`

6 Click OK.

7 Click the Save button.



You have now created the business object. Now validate it by clicking the Validate button.



1.2 Create QAD QXtend Inbound Profile

Once the business object has been created, you must create a profile that can be used to update the target system/domains in the data synchronization process. This requires a new profile and mapping of the names that are used to create the XML passed to QXI.

When you create a business object in QXO it creates a default profile. You should never make changes to the default profile as each time the business object changes it regenerates the profile and you will lose all of your changes. The first thing you should do is copy the profile to create a new one that you can make changes to.

Create the New Profile

1. Open the QXO Web application in Internet Explorer `http://qaddemo:8080/qxo`
2. Select the Profiles tab.
3. Ensure that the App Type is set to `QADEE`.
4. Click the fetch button to get all profiles.
5. Select the `GenCodes` profile under the GenCodes business object.
6. Click the Copy button.
7. Enter the following details:

- Name = `maintainGeneralizedCode`
- Type = Data Synchronization
- QDoc Name = `maintainGeneralizedCode`
- QDoc Version = `ERP3_1`

8 Click the Save button.

Map the Business Object Name to the QXI QDoc Names

1 The profile is used to map the business object to the QXI QDoc structure and names. The following steps will walk you through this process.

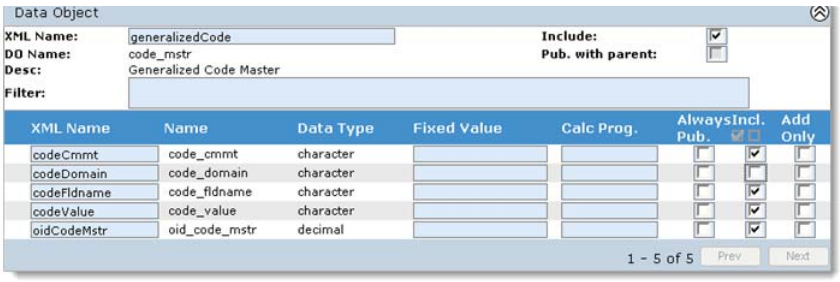
2 Drill down to the `code_mstr` data object on the profile menu.

3 Click the Edit button.

4 Change the following values:

a Data Object XML Name = `generalizedCode`

b Do not include `code_domain`, so clear the Include check box.



5 Click the Save button.

1.3 Configure QAD QXtend Outbound Services

The QXO services are currently configured to use the standard Generalized Code business object that is shipped with QAD QXtend. You need to change the configuration of these services and the subscribers so that they use the business object and profile created in the previous steps.

1.3.1 Message Publisher

1 Open the QXO Web application in Internet Explorer: `http://qaddemo:8080/qxo`.

2 Select the Configuration tab.

3 Select the Message Publisher menu item.

4 Select the MP1 menu item in the menu tree.

5 Click the Lookup button for the Registered Business Objects.

6 Ensure that the App Type is set to `QADEE`.

7 Click the fetch button to get all business objects.

8 Remove the QADEE/GeneralizedCode business object.

9 Add the QADEE/GenCodes business object.

10 Click the OK button.

11 Click the Save button.

1.3.2 Subscribers

For each of the subscribers defined in QXO you need to remove the GeneralizedCode Sync profile and replace it with the maintainGeneralizedCode profile.

1 Open the QXO Web application in Internet Explorer: `http://qaddemo:8080/qxo`.

2 Select the Configuration tab.

3 Select the Subscribers menu item.

4 Select the subscriber to update from the menu. (You will update each subscriber in turn.)

5 Click the Register Profiles button.

6 Ensure that the App Type is set to `QADEE`.

7 Click the fetch button to get all profiles.

8 Remove the QADEE/GeneralizedCode/MaintainGeneralizedCode profile.

9 Add the QADEE/GenCodes/maintainGeneralizedCode profile.

10 Click the OK button.

11 Click the Save button.

12 Repeat Steps Select the subscriber to update from the menu. (You will update each subscriber in turn.)—Click the Save button for each of the profiles.

1.4 Process Transactions

The configuration of QXO is now complete for the new Generalized Code business object and profile. Now you must stop and start the services and process some transactions from the master domain.

1 Stop all of the QXO services, using the Dashboard, Services menu.

2 Start all of the QXO services, using the Dashboard, Services menu.

Log in to the QAD EE application on the Windows client and process some Generalized Code transactions in the 10USA domain. Check in QXO that messages are being created for the target domains and that the requests are being processed correctly. Also verify that the target domains are updated.

2. Sales Order File Generation

The business objects in QXO allow you to build complex objects that have many levels and relationships. In this lab exercise you will build a Sales Order business object and then write it out to an XML file. The Sales Order object will need to contain information from the Sales Order master and detail tables.

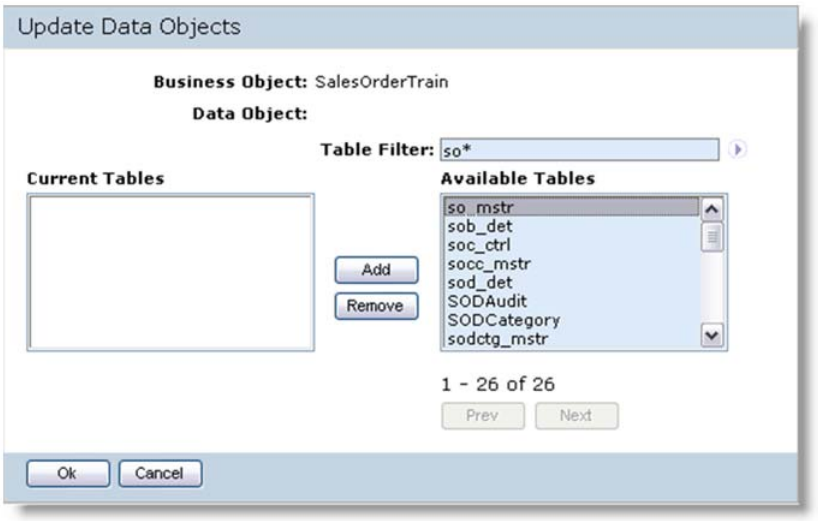
2.1 Create New Business Object

First create a new business object that specifies the data that needs to be extracted.

1. Open the QXO Web application in Internet Explorer: `http://qaddemo:8080/qxo`.
2. Select the Business Objects tab.
3. Ensure that the App Type is set to QADEE.
4. Click the New button.
5. Click the No button for “Create a direct data publish business object?”
6. Enter the following details for the Business Object: Name = `SalesOrderTrain` and Description = `Sales Order Business object created for QAD QXtend Training`.
7. Click the Save button.

Add Tables to the Business Object

1. Select the SalesOrderTrain business object from the Business Objects menu.
2. Click the Edit button.
3. Click the Tables button.
4. Enter `so*` into the Table Filter and click the Fetch button.



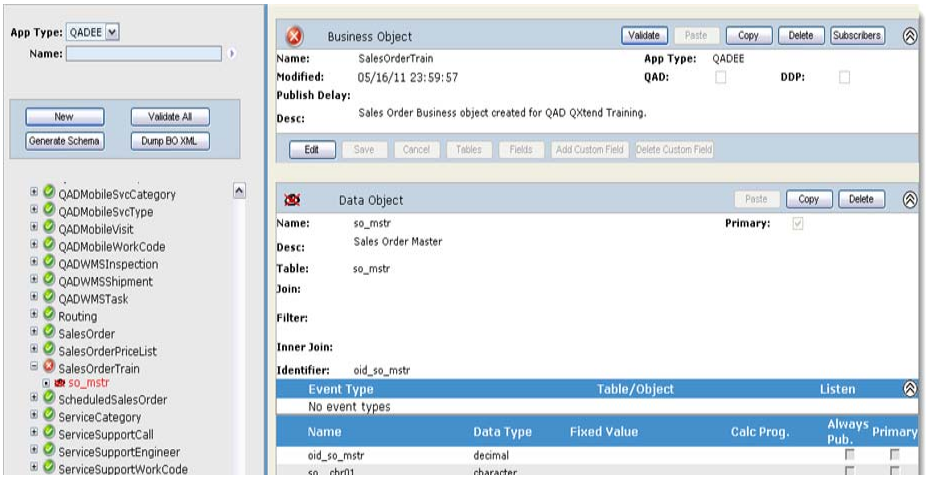
5. Select the `so_mstr` table and click the Add button.
6. Click the OK button.

7 Click the Save button.

Edit the Fields

1 Select the SalesOrderTrain business object from the Business Objects menu.

2 Drill down to the `so_mstr` data object.



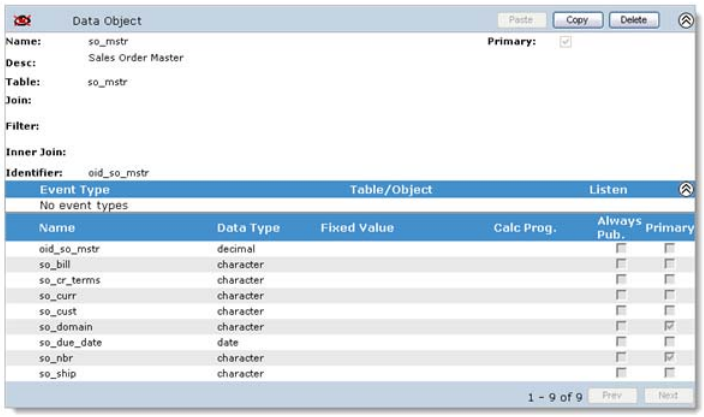
3 Click the Edit button.

4 Click the Fields button.

5 Ensure that only the following fields are included:

- oid_so_mstr
- so_bill
- so_cr_terms
- so_curr
- so_cust
- so_domain
- so_due_date
- so_nbr
- so_ship

6 Click the Save button.



Add the Sales Order Detail as a Child to the Sales Order Master Table

1 Select the SalesOrderTrain business object from the Business Objects menu.

2 Drill down to the `so_mstr` data object.

3 Click the Edit button.

4 Click the Tables button.

5 In the Table Filter field, enter `so*` and click the Fetch button.

6 Add the `sod_det` table. This will add `sod_det` as a child of the `so_mstr` table.

7 Click the OK button.

8 Click the Save button.

Edit the Join between so_mstr and sod_det and the Fields Included from the sod_det Table

1 Select the SalesOrderTrain business object from the Business Objects menu.

2 Drill down to the `sod_det` data object.

3 Click the Edit button.

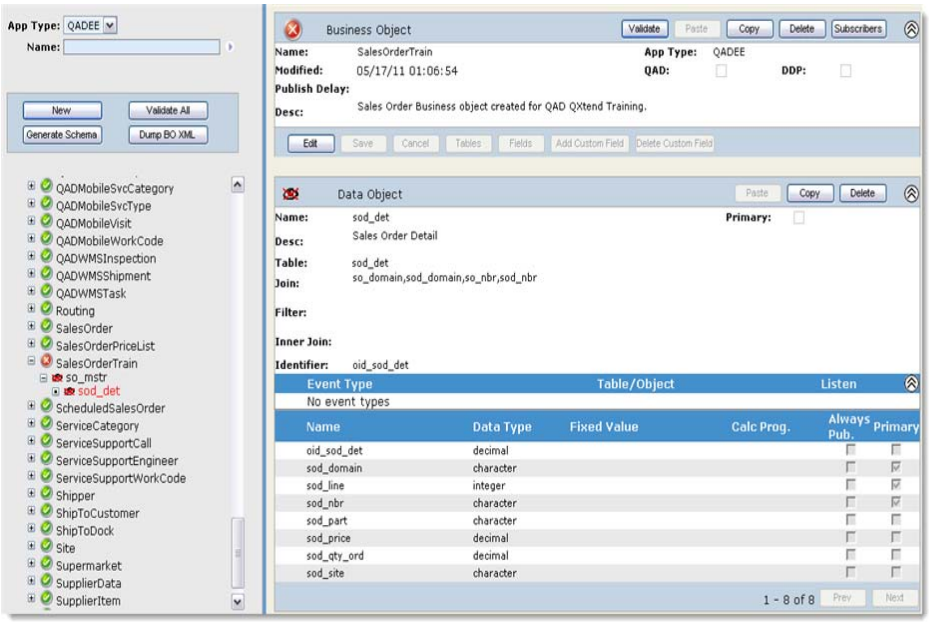
4 Set Join = `so_domain,sod_domain,so_nbr,sod_nbr`.

5 Click the Fields button.

6 Ensure that only the following fields are included

- oid_sod_det
- sod_domain
- sod_line
- sod_nbr
- sod_part
- sod_price
- sod_qty_ord

- sod_site



7 Click the Save button.

8 You have created the business object. Now validate the business object by clicking the Validate button.



2.2 Configure QAD QXtend Outbound Services

You need to change the configuration of the services and the subscribers so that they publish the new Sales Order object created in the previous section.

2.2.1 Message Publisher

1 Open the QXO Web application in Internet Explorer: `http://qaddemo:8080/qxo`.

2 Select the Configuration tab.

3 Select the Message Publisher menu item.

4 Select the MP1 menu item from the menu tree.

5 Click the Lookup button for the Registered Business Objects.

6 Ensure that the App Type is set to `QADEE`.

7 Click the fetch button to get all business objects.

8 Add the QADEE/SalesOrderTrain business object.

9 Click the OK button.

10 Click the Save button.

2.2.2 Subscribers

You want to create an XML file that contains the profile data. To do this you need to change the configuration of the FileDrop subscriber.

1 Open the QXO Web application in Internet Explorer: `http://qaddemo:8080/qxo`.

2 Select the Configuration tab.

3 Select the Subscribers menu item.

4 Select FileDrop from the menu.

5 Click the Register Profiles button.

6 Ensure that the App Type is set to QADEE.

7 Click the fetch button to get all Profiles.

8 Add the QADEE/SalesOrderTrain/SalesOrderTrain profile.

9 Click the OK button.

10 Click the Save button.

2.3 Enable so_mstr Events in QXO

The configuration of QXO requires that the event types for tables being replicated must be enabled. In the current configuration the database trigger is not enabled for `so_mstr`; you need to enable the event type before testing the synchronization process.

1 Open the QXO Web application in Internet Explorer: `http://qaddemo:8080/qxo`.

2 Select the Configuration tab.

3 Select the Source Applications menu item.

4 Select QADEE from the menu.

5 Select QADERP from the menu.

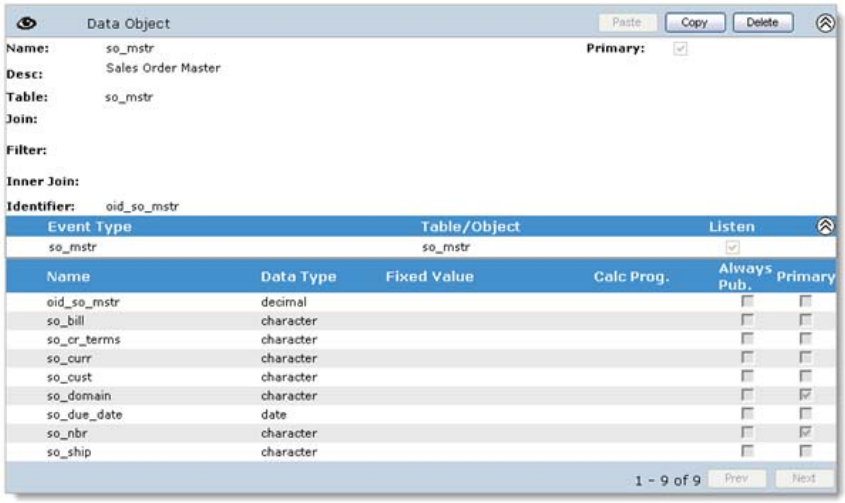
6 Click Event Types from the menu.

7 Find the `so_mstr` event type and select the Active check box.

8 In the Data Object Listening dialog, select the Listen check box for SalesOrderTrain/so_mstr.

9 Click the OK button.

10 Click the Save button.



2.4 Process Transactions

The configuration of QXO is now complete for the new Sales Order business object and profile. Now we will stop and start the services, and process some transactions from the master domain.

1 Stop all of the QXO services, using the Dashboard, Services menu.

2 Start all of the QXO services, using the Dashboard, Services menu.

Log in to the QAD EE application on the Windows client, update an existing order, and create a new order in the 10USA domain. Check in QXO that messages are being created for the FileDrop subscriber. Check the directory configured for the subscriber to see that the files are being created.

3. Synchronize Item Master Data

In the first part of this lab, you created a data synchronization configuration, creating a new business object and profile. This section describes using the business objects and profiles that are shipped as part of the standard product.

In this scenario we want to synchronize Item Master data from the 10USA domain into the 11CAN domain. QXO comes with an Item business object that has a MaintainItemMaster profile that is built to be used with QXI. This means that you do not need to create any business objects for this lab. However, we cannot use the existing MaintainItemMaster profile because it is for SIAPI (we will learn SIAPI later). In this lab we create a copy of the MaintainItemMaster profile to load item data into the target domain through UIAPI.

3.1 Create Profile

1 Open the QXO Web application in Internet Explorer: `http://qaddemo:8080/qxo`

2 Select the Profiles tab.

3 Select the Item business object.

4 Select the MaintainItemMaster menu item.

5 Click the Copy button.

6 Set the following values: Name = `ItemSyncUIAPI` and QDoc Version = `ERP3_7`.

7 Click the Save button.

3.2 Configure QAD QXtend Outbound Services

You need to change the configuration of the services and the subscribers so that they publish the Item data.

3.2.1 Message Publisher

1 Open the QXO web application in Internet Explorer http://qaddemo:8080/qxo

2 Select the Configuration tab.

3 Select the Message Publisher menu item.

4 Select the MP1 menu item from the menu tree.

5 Click the Lookup button for the Registered Business Objects.

6 Ensure that the App Type is set to `QADEE`.

7 Click the fetch button to get all business objects.

8 Add the QADEE/Item business object.

9 Click the OK button.

10 Click the Save button.

3.2.2 Subscribers

You want to synchronize data from the 10USA domain to the 11CAN domain. To do this you need to add the profile to the relevant subscriber.

Update the 11CAN Subscriber

1 Open the QXO Web application in Internet Explorer: `http://qaddemo:8080/qxo`.

2 Select the Configuration tab

3 Select the Subscribers menu item.

4 Select QMSEMEA from the menu.

5 Click the Register Profiles button.

6 Ensure that the App Type is set to `QADEE`.

7 Click the fetch button to get all Profiles.

8 Add the QADEE/Item/ItemSyncUIAPI profile.

9 Click the OK button.

10 Click the Save button.

Set the Fixed Value for Site field.

1. In the laft panel, select the ItemSyncUIAPI profile under subscriber 11CAN.
2. In Subscrber Profile Configuration Parameters panel, set the Fixed Value for @Site@ to 11-100.
3. Click the Save button.

3.3 Enable ItemMaintenance Events in QXO

The configuration of QXO requires that the database triggers for tables being replicated or business events for a maintenance program must be enabled. In the current configuration they are not enabled for item master; you need to enable an event before testing the synchronization process.

1. Open the QXO Web application in Internet Explorer: `http://qaddemo:8080/qxo`.
2. Select the Configuration tab.
3. Select the Source Applications menu item.
4. Select QADEE from the menu.
5. Select QADERP from the menu.
6. Click Event Types from the menu.
7. Find the ItemMaintenance event type in Business Events and select the Active check box.
8. Click the Save button.

3.4 Process Transactions

The configuration of QXO is now complete for the new Sales Order business object and profile and all that remains is for you to stop and start the services and process some transactions from the master domain.

1. Stop all of the QXO services using the Dashboard|Services menu.
2. Start all of the QXO services using the Dashboard|Services menu.

3.4.1 Create New Item

1. Log in to the QAD EE Application on the Windows Client.
2. Create a new Item and ensure you set the following values:
   - Product Line = `10`
   - Status = `ACTIVE`
   - ABC Class = `A`
   - Location = `010`

Validate that the item has been processed by checking the subscriber messages in QXO. The item will have been loaded into target domain and should have a DLV status.

3.4.2 Modify Item

Modify the Item you just created and ensure you set the following values:

- ABC Class = `B`

Validate that the item has been processed by checking the subscriber messages in QXO. The item will have been changed in target domain and should have a DLV status.

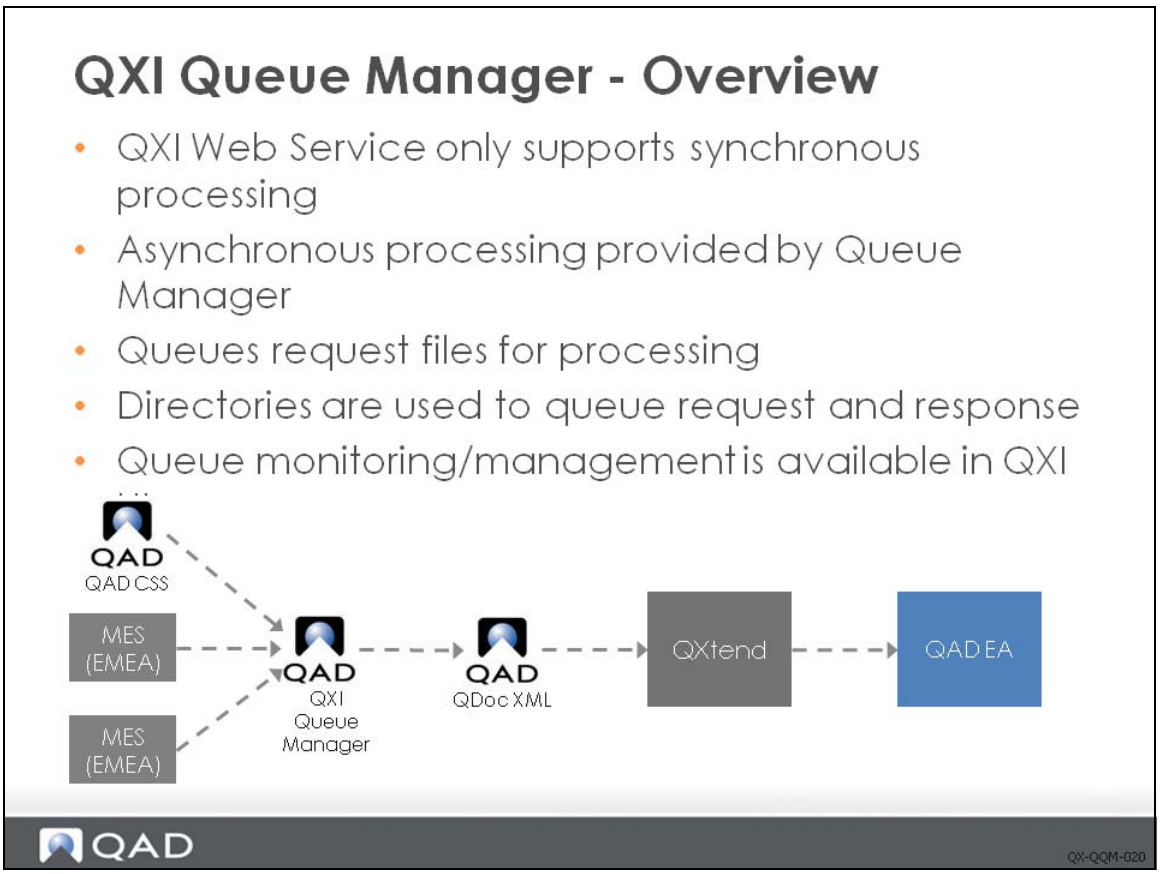
3.4.3 Delete Item

Delete the Item you just modified. Validate that the item has been processed by checking the subscriber messages in QXO. The item will have been deleted in target domain and should have a DLV status.

Chapter 7

QXI Queue Manager

Overview



The Web service provided by QXI only supports synchronous requests; when a client makes a request to the Web service, the client must stay connected until a response message is returned. The client cannot submit a request and have the response delivered later. Not all applications can process synchronous messages. To remedy this, QXtend has an asynchronous messaging solution called the Queue Manager as part of the QXI suite.

While various messaging products—for example, Sonic, IBM WebSphere, and WebMethods—support asynchronous messaging, these products are expensive, but do have advanced queuing capabilities. The QXtend Queue Manager is a basic queueing application that allows operating system directories to be treated as request and response queues.

The Queue Manager allows the creation, monitoring, and management of multiple queues. Each queue consists of:

- A request directory used to queue requests that need to be sent to the QXI web service
- A response directory used to store responses to the requests that have been processed by the queue.

Request XML messages with or without a SOAP envelope are dropped into a file in the queue's request directory with a `.req` file extension. The directory is polled and any request files are delivered to the QXI Web service via a synchronous call. The response from the call and the original request file are then moved to the response's directory for analysis and, if necessary,

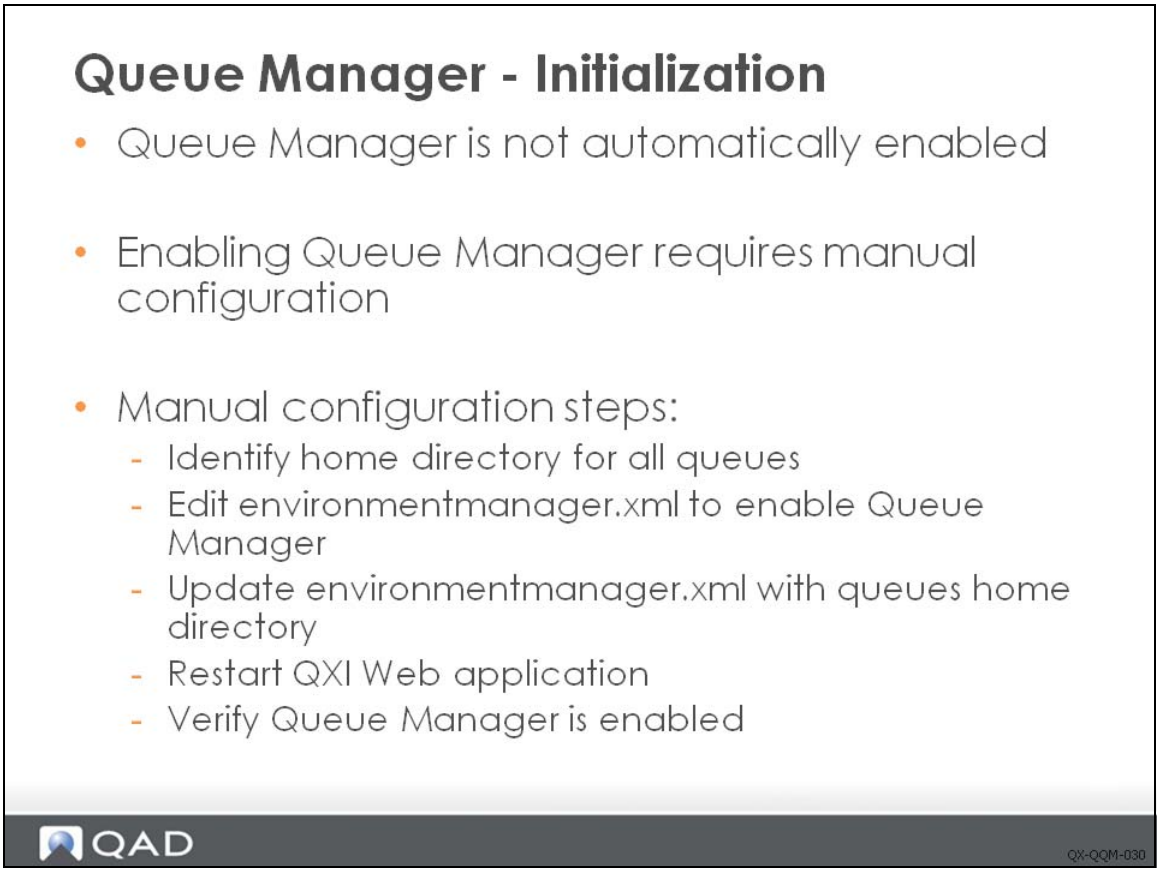
further processing by the external application. The processing of response messages by the external application is not part of QXtend functionality; if this is required, it must be implemented as part of integration development.

A management and monitoring application UI is provided as part of QXI. This UI allows you to:

- Create new queues
- Update existing queue configuration
- View queue contents
- View requests/responses
- Delete processed messages
- Edit and resubmit failed messages

When placing requests into a queue, you should create the file with a `.tmp` extension. After creating the file, rename it to `.req`. Changing the extension prevents the request file from being picked up for processing by the queue before it has been completely written to disk.

Initializing the Queue Manager



The Queue Manager is not automatically enabled when QXtend is installed—you must initialize it manually. To do this you edit a QXI Web application configuration file. You will need appropriate file permissions to save the changes you make to the file.

To enable the Queue Manager, do the following:

1 Identify Queue Directory. The Queue Manager manages a group of queues. All queues must be located in the same parent directory. When queues are added they are automatically created in this directory. During installation QXI creates a default directory that is home for all queues:

`<tomcat-home>/webapps/<qxi-webapp>/qxtendQueues`

This directory should be used to host all queues. If you do not want to use this directory, you can specify another directory; see *Technical Reference: QAD QXtend* for details.

2 Update `environmentmanager.xml`

a Open `environmentmanager.xml` in:

`<TOMCAT_HOME>/webapps/<qxi-webapp>/WEB-INF/conf`

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

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

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

d 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, you must specify the full path in `environmentmanager.xml` otherwise the Queue Manager will not work correctly. For example:

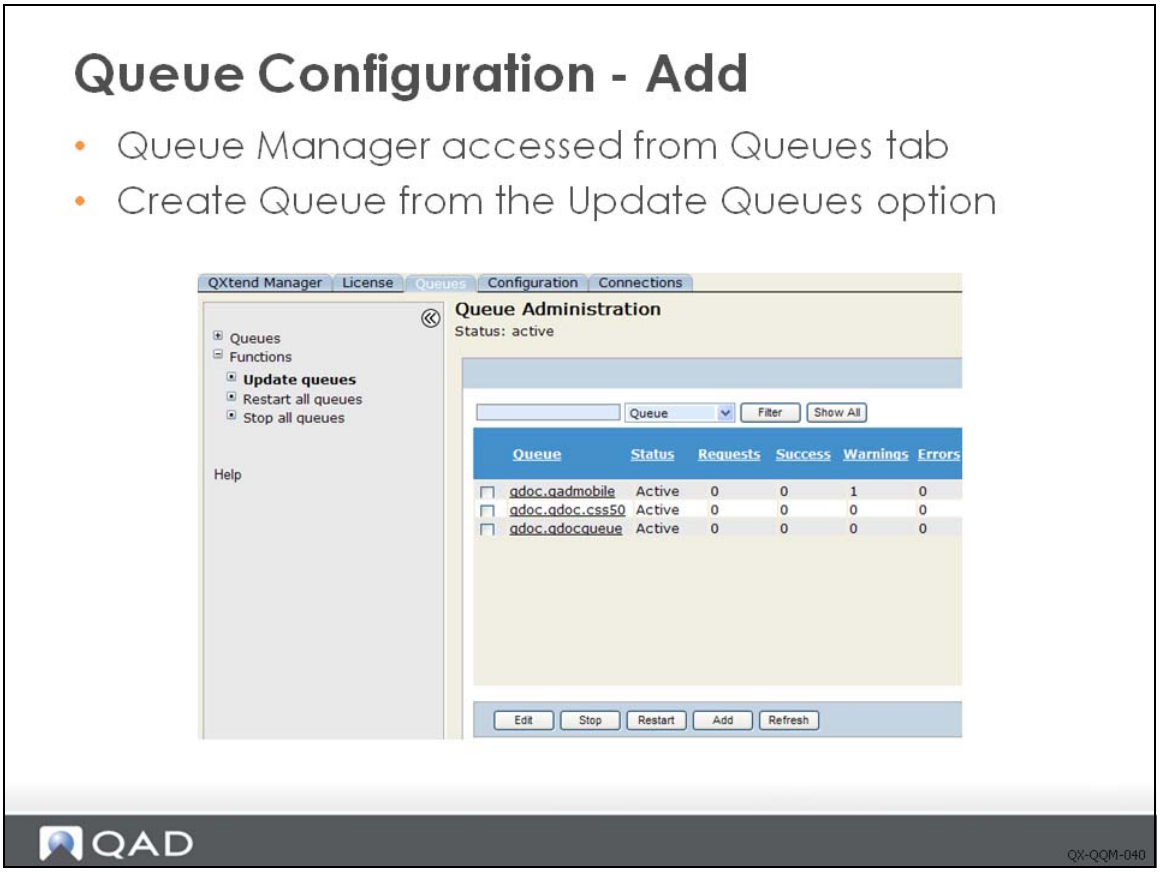
```
<manager class=
"com.qad.qxtend.queue.directory.DirectoryManager" Param=
"c:\Tomcat5.5\webapps\qxtendserver\qxtendQueues"/>
```

e Save the file.

3 Restart QXI Web Application. The changes you make to the configuration file are not automatically applied when the file has been saved. For the changes to take effect you must restart QXI. The best way to do this without stopping and starting the entire Tomcat instance is to use the Start and Stop options on the Tomcat Manager page. You can access this from the base Tomcat URL `http:\\<tomcat-machine>:<tomcat-port>`.

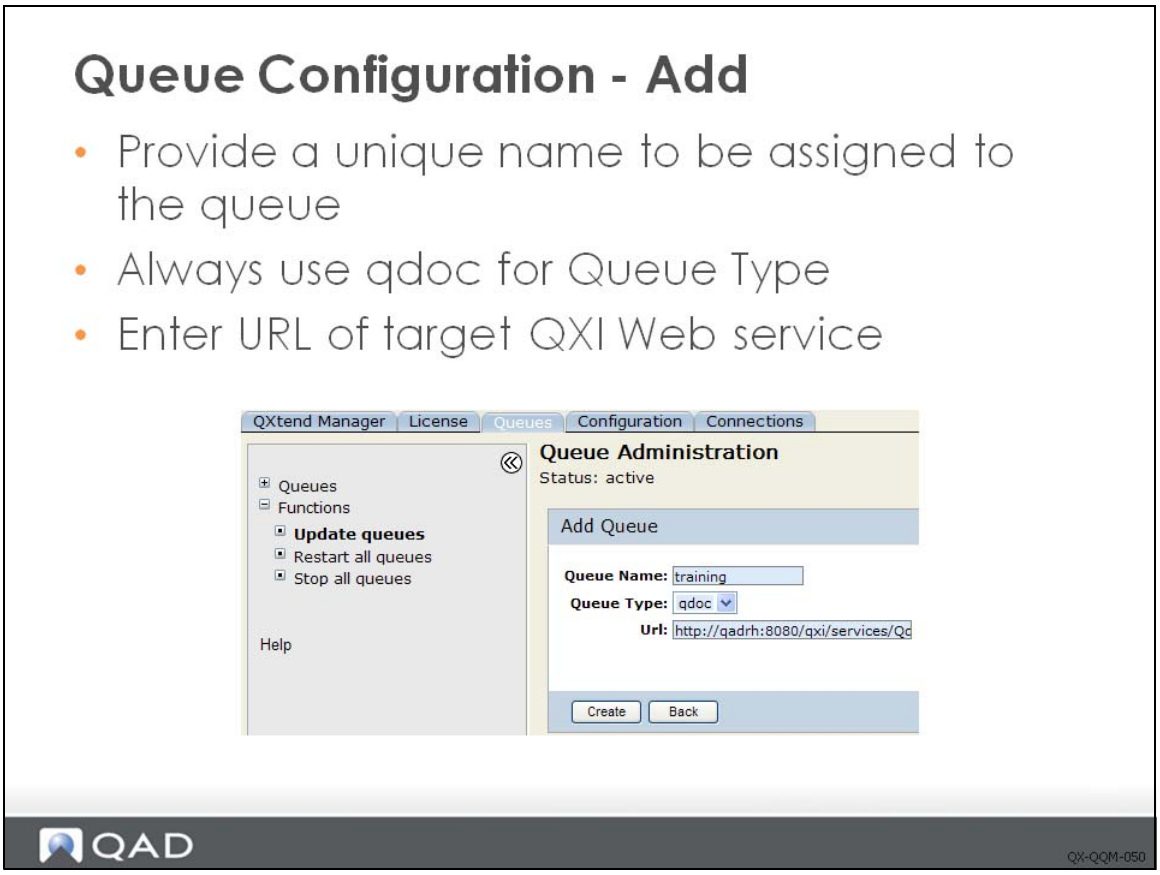
4 Verify the Queue Manager. Once QXI is restarted, you can verify that the changes have been applied by opening QAD QXtend. If the changes have been applied correctly, the Queues tab displays on the tab bar.

Adding a Queue



After the Queue Manager has been enabled, the Queues tab displays in the QXI instance. The Queues tab contains the user functions for managing and monitoring queues.

The Functions node on the tree menu contains all queue management functions, including the Add option.



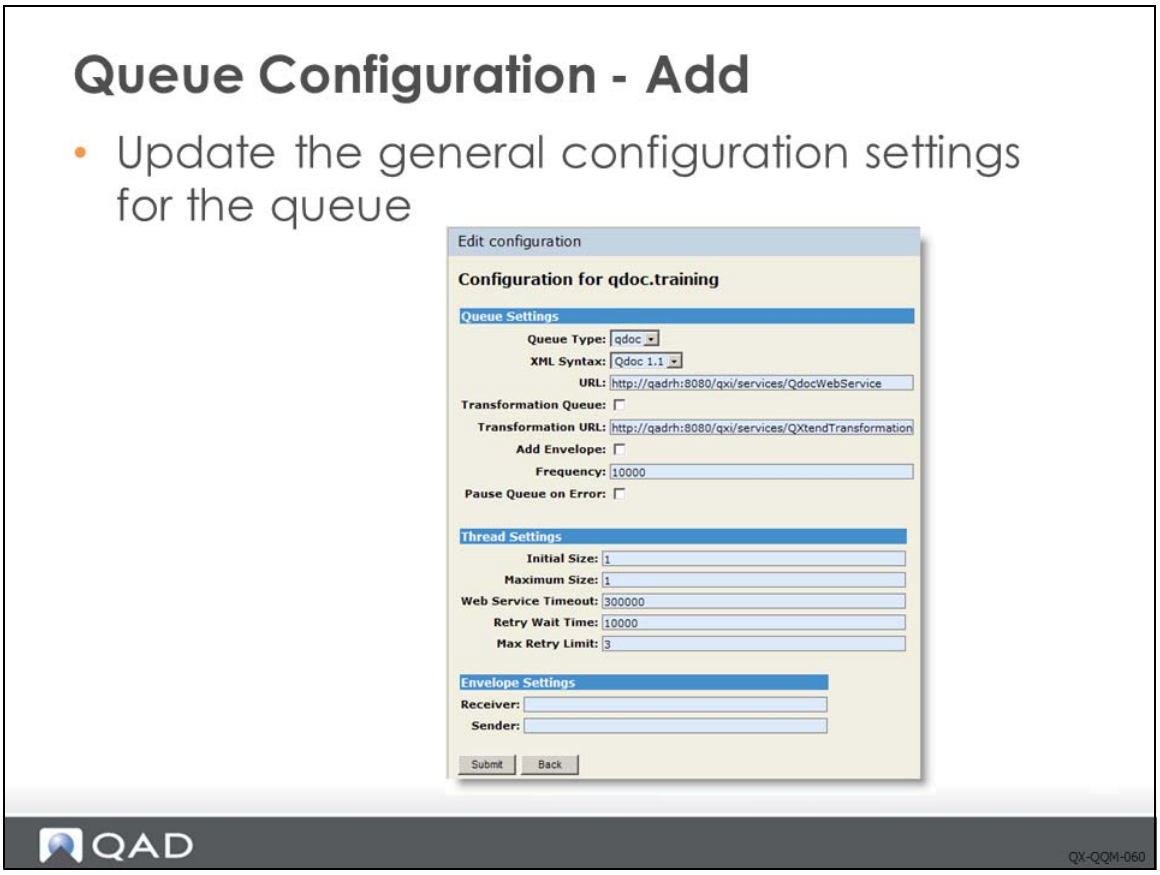
To create a new queue, select the Functions node on the tree menu on the Queues tab. Select the Update queues option, then click Create.

The initial creation of a queue only requires completing three fields:

Queue Name. Enter a unique name within the Queue Manager instance. This name identifies the queue and also is the name of the directory that gets created in the Queue Manager directory.

Queue Type. This should always be qdoc.

URL. The URL to send the requests to. You can post messages to any instance of QXI from the queue. The URL entered here identifies where messages are sent.



Configuration settings control a queue's behavior. When you create a queue, it is configured automatically using default settings—you can change these if required at any time. When you create a queue, the Edit Configuration screen displays. The screen contains the following fields:

Queue Settings

XML Syntax. The syntax of the QDoc messages processed by this queue. New implementations should use the QDoc 1.1 syntax. Support for the QDoc 1.0 syntax is provided only for backward compatibility.

Note This course does not describe settings for the 1.0 syntax.

URL. The location of the QXI Web service that will process the request files processed by this queue.

Transformation Parameters. These parameters are not discussed as the feature is rarely implemented. These parameters will be removed from the product and are an advanced feature.

Add Envelope. All requests to QXI must include a valid QDoc SOAP envelope. If the incoming requests do not have a SOAP envelope, set this to True; the Queue Manager adds a SOAP envelope to each incoming message before sending it to QXI. If Add Envelope is selected, you also must enter values for envelope settings.

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

Pause Queue on Error. Specify how Queue Manager handles queues when encountering errors. If this is set to yes, when error occurs this queue will be paused. When restarted, Queue Manager resumes processing the QDoc that paused the queue. If this is set to no, when error occurs, Queue Manager proceeds to the next QDoc without pausing the queue.

Thread Settings

Configure the queue as either a single or multi-threaded queue. If high volumes of messages are expected for a queue, in order to prevent a significant message backlog, you can assign multiple processing threads to a queue enabling the queue to process multiple messages simultaneously.

Initial Size. The number of threads to start initially when the queue is started. The value entered must be less than the Maximum Size parameter.

Maximum Size. The maximum number of threads (or QDocs) that can be processed at a time. Since one thread consumes one agent in the QXI connection pool when processing a request, make sure that the queue and connection pool configuration are complementary.

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

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

Max Retry Limit. The maximum number of times Queue Manager will try to send a request QDoc before it stops trying. This allows enough time for web services to get up and running.

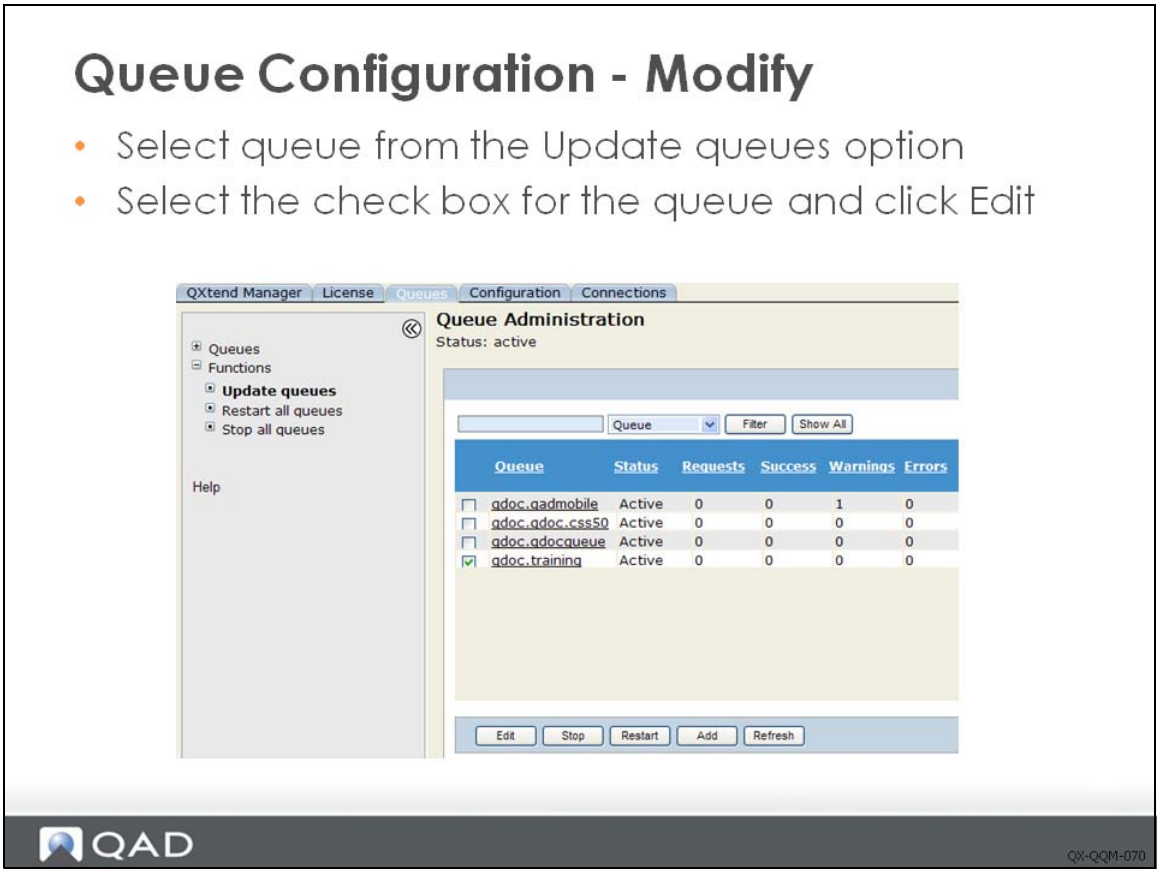
Envelope Settings

If the Add Envelope option is selected, you must provide appropriate envelope settings:

Receiver. Identifies the QDoc recipient, which is by definition an QAD EA instance. Enter the name of a receiver defined in QXI.

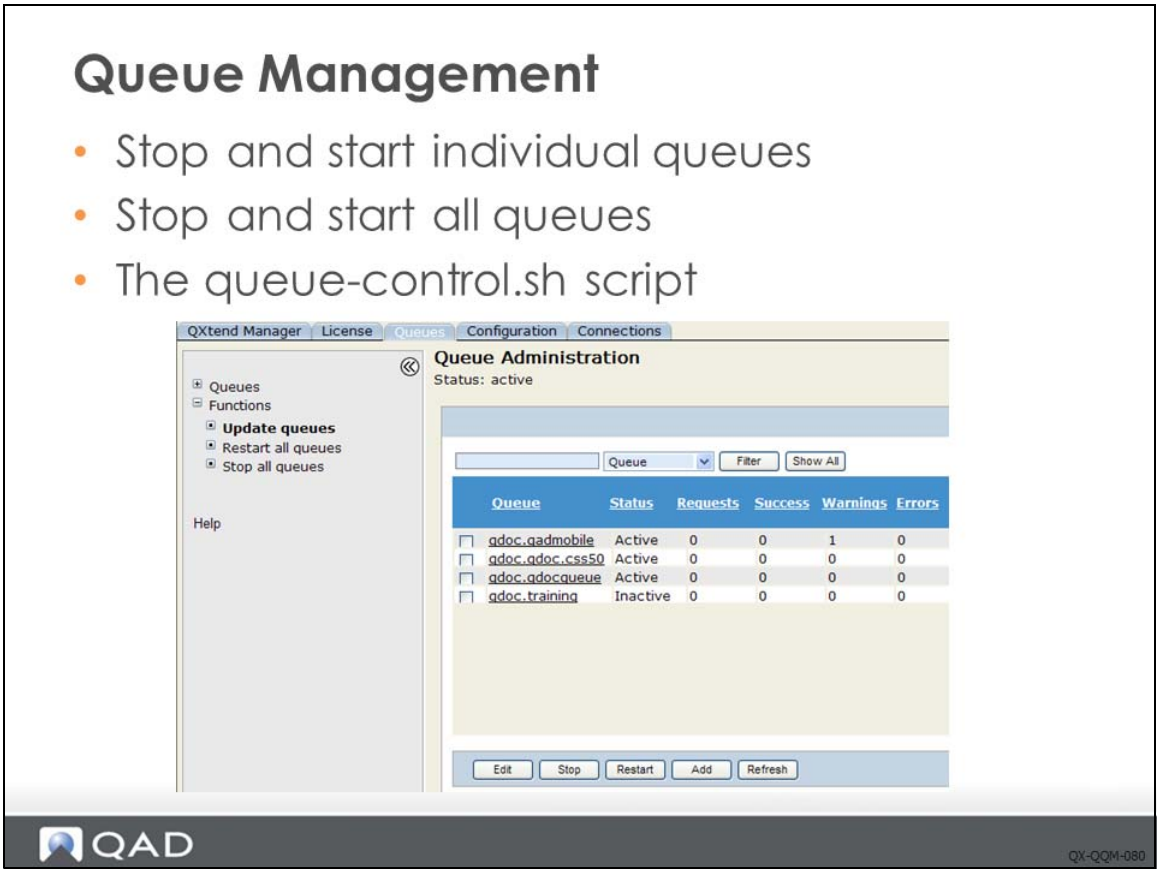
Sender. Identifies the QDoc source application.

Modifying a Queue



You can edit the queue configuration any time: navigate to the Update queues node on the tree menu, select the queue to update from the list, then click Edit. The Edit Configuration screen displays with the current queue configuration. Modify queue settings as required.

Queue Management

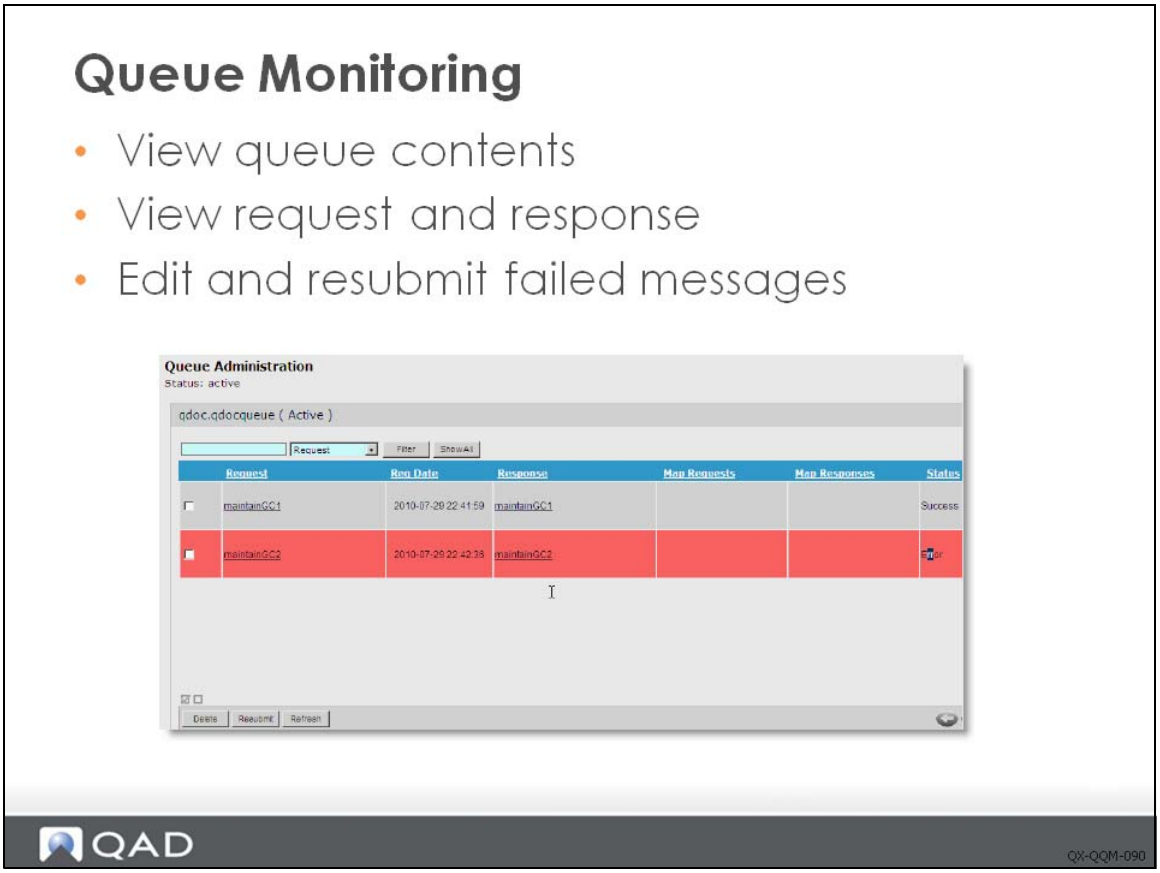


The Queue Manager is responsible for processing messages and monitoring queues for messages that require processing. The list of queues displayed in the Update queues node in the tree menu shows the status of the current queue: Active or Inactive. Inactive queues have been stopped and are not processing messages delivered to the queue. To stop or start queues, select the check box next to the queue and click Stop or Restart as required.

To stop or start all queues, use the Stop all queues and Restart all queues options on the tree menu.

You can also use the `queue-control.sh` script to start, stop and restart queues, as well as query the connection pool status. This script can be found under `WEB-INF/scripts` directory of QXI web application.

Queue Monitoring



You can monitor the queues managed by the Queue Manager by using the Queues tab in QXI. Select the Queues node in the tree menu to display all queues currently being monitored by the Queue Manager. Select the queue you want to monitor from the list of queues displayed.

When you choose a queue name on the tree menu — or select a queue from Functions|Update Queues — the contents of the queue displays. The queue document list is color coded:

- Red lines are documents that returned an error.
- Yellow lines are messages that processed but a warning was returned.
- Green lines are messages that are either pending or in process.
- Grey lines are successfully processed message.

By default, the display shows 100 records. Use the scroll bar on the queue to view all 100 documents. You also can set the number of documents to display by entering a value in the Records field, and specify the message to start from by using the Start From field.

Sorting

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

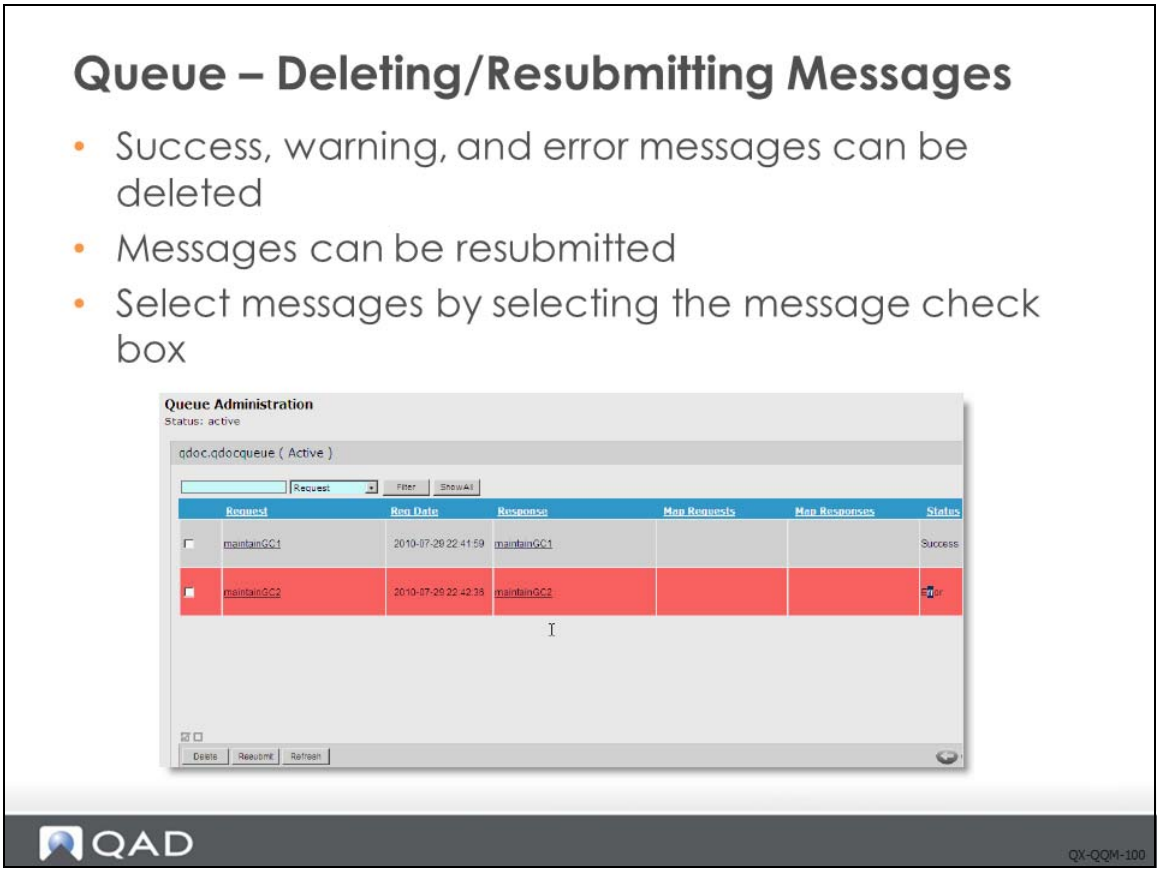
Filtering

You can filter the queue by entering criteria text in the field above the table. Then select the document type from the drop-down list. Choose Filter to select the 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 warnings present in all queues that contain the character string `test` in the name. Click Show All to display all queues again.

Note To maintain optimal performance, filter and sort options do not refresh the data — they filter data already displayed. To refresh the data with changes to the server, click Refresh before sorting or filtering.

Deleting Messages

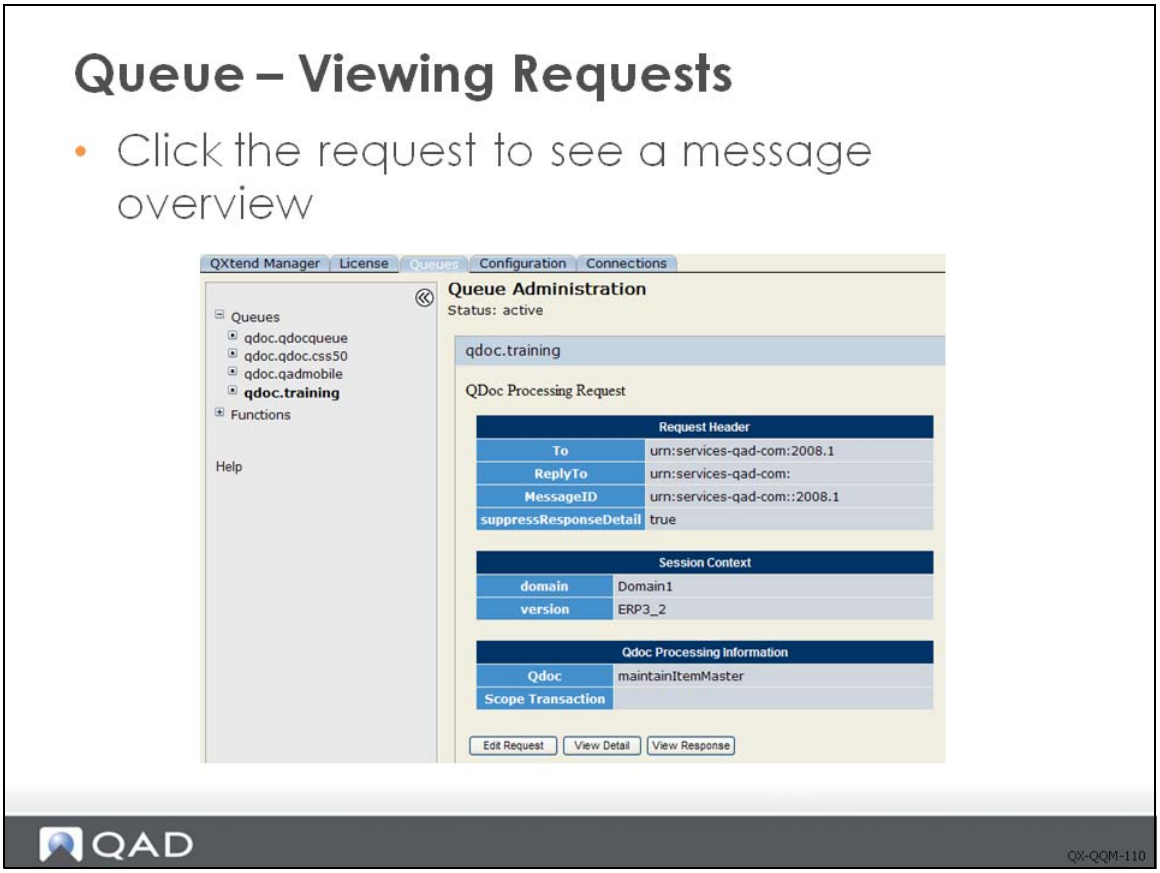


Messages processed by the Queue Manager are not purged automatically—you can delete messages from the Queue Manager. Select the queue you want to manage, select the messages you want to delete from the system by selecting the check box next to the messages in the queue, then click Delete. Only messages with a Success, Warning, or Error status can be deleted.

Note If you intend to process a high volume of messages through any of the queues managed by the Queue Manager, you should set up an automated clean-up routine to delete old messages. If old messages are not deleted regularly, the response directory becomes too large and is then difficult to work with; in addition, the queue might contain thousands of messages, making it difficult to find and manage messages.

You can select and resubmit failed QDocs directly using the Resubmit button in Queue Administration. You can also first edit a failed QDoc to correct any errors in the document and then resubmit it in the Edit QDoc Request screen, which will be introduced later.

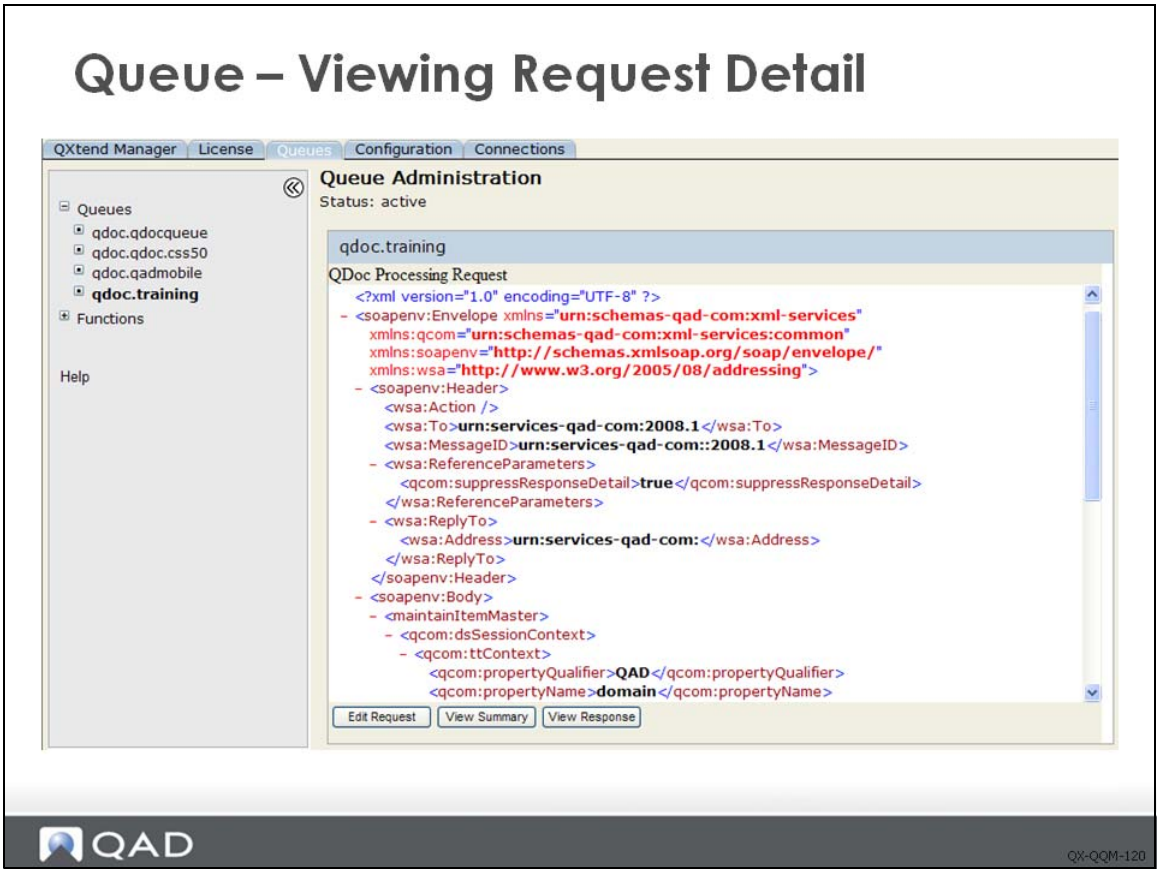
Viewing Requests/Responses



You can drill into any request in the current queue being monitored to view the request's contents. To view a request, click the name of the request message in the queue. The name in the request column is a link that displays the message overview screen. From the message overview screen you can view the detail (actual XML) of the request and to drill into the Response message overview screen.

If the response message has an Error status, you can edit the request and resubmit it.

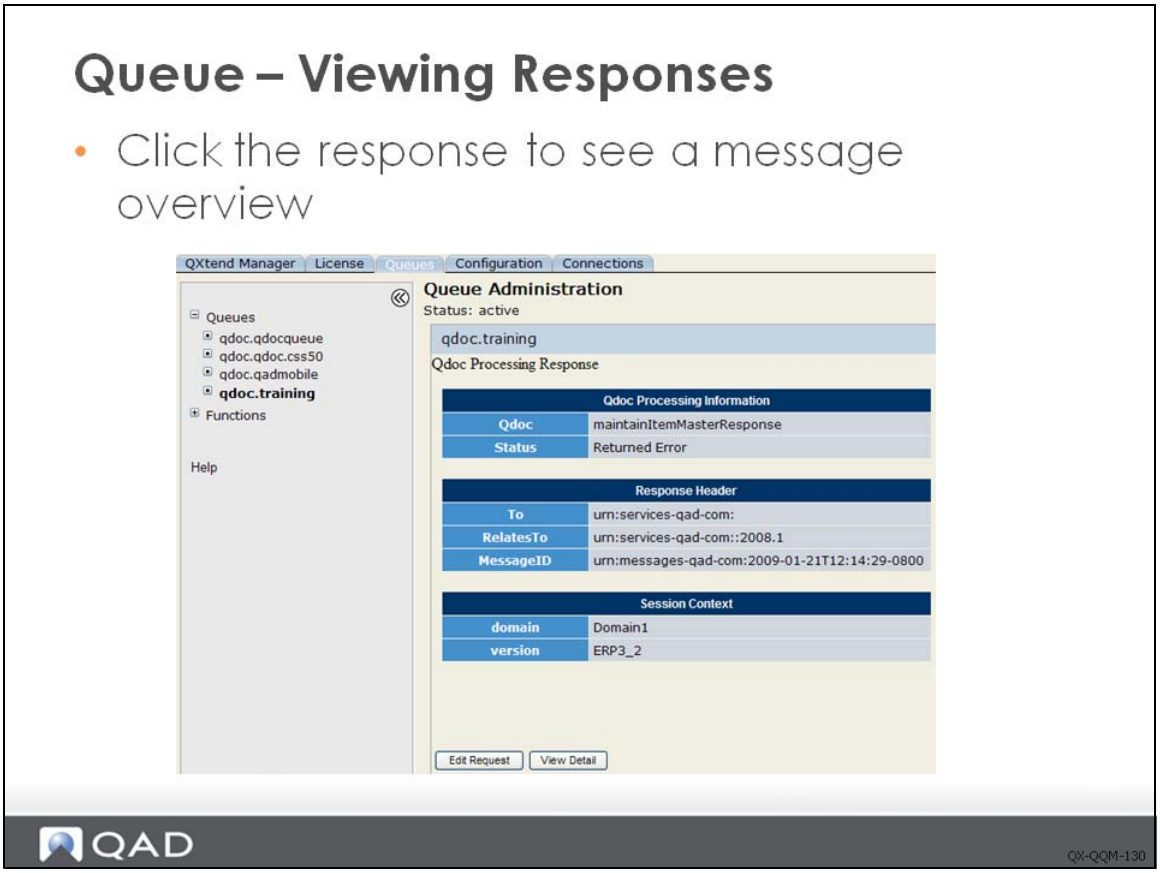
Viewing Request Details



From the request overview screen you can click View Detail to display the actual XML message that was submitted to the Queue, and subsequently to the QXI Web service. From this screen you can switch back to the detail view or drill into the Response overview screen.

If the response message has an Error status, you can edit the request and resubmit it.

Viewing Responses

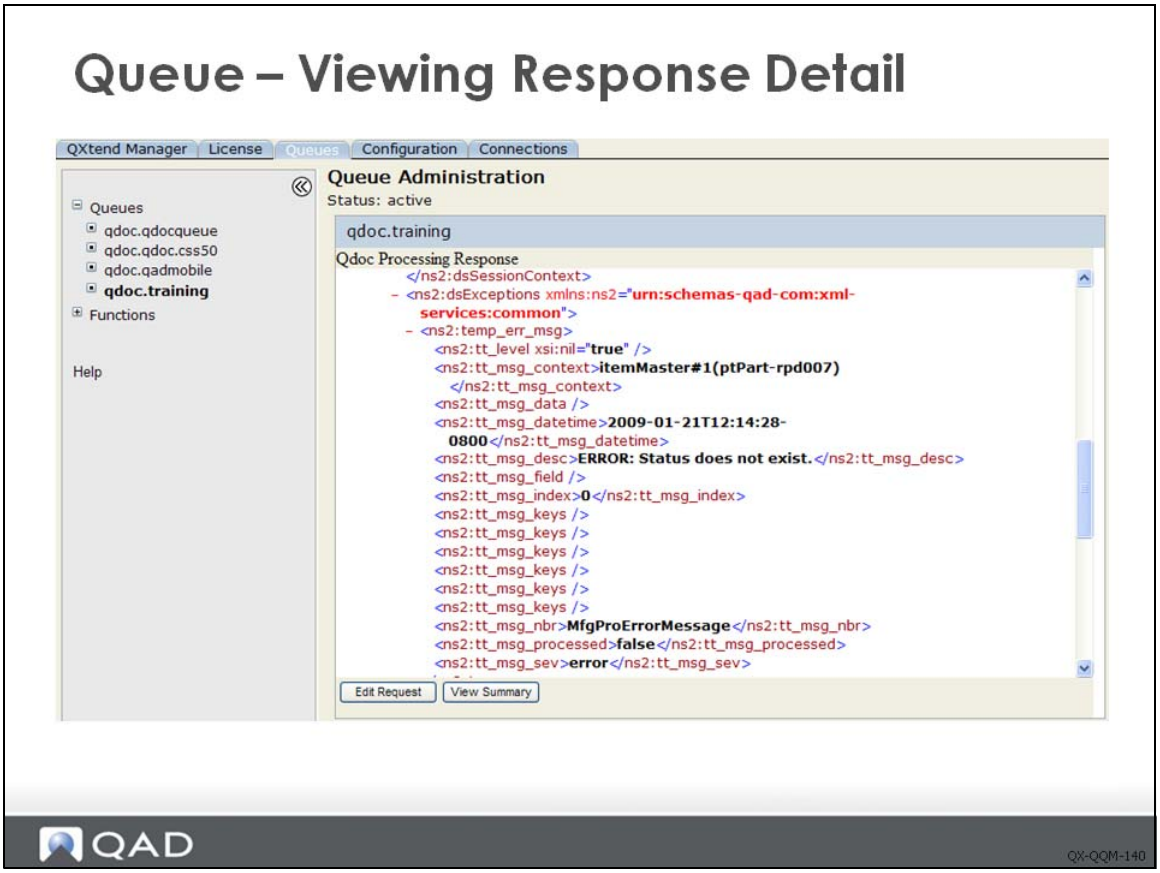


You can drill into any response in the queue currently being monitored to view the response's contents.

To view a response, click the name of the response message in the queue. The name in the response column is a link that displays the message overview screen. From the message overview screen you can view the detail (actual XML) of the response.

If the response message has an Error status, you can edit the request and resubmit it.

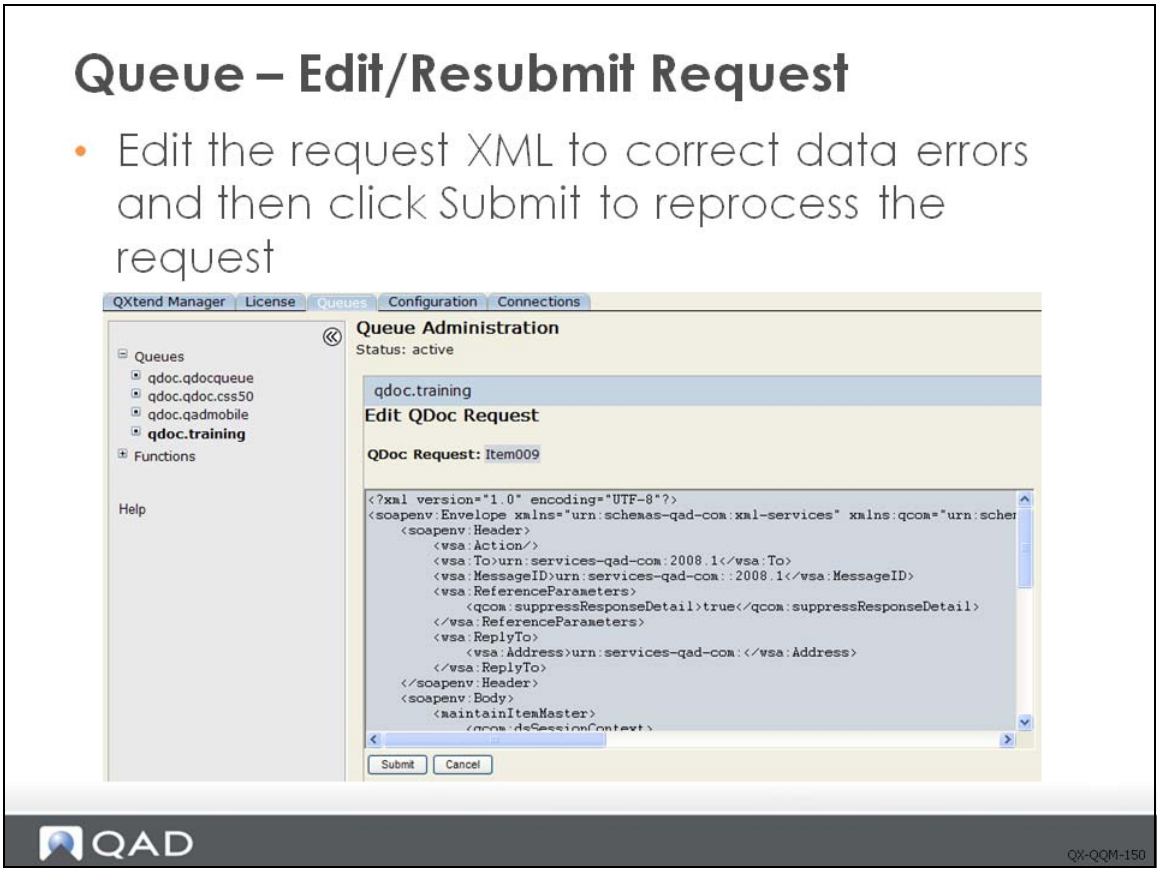
Viewing Response Details



From the response overview screen you can click View Detail to display the actual XML message that was submitted to the Queue, and subsequently to the QXI Web service. From this screen you can switch back to the detail view.

If the response message has an Error status, you can edit the request and resubmit it.

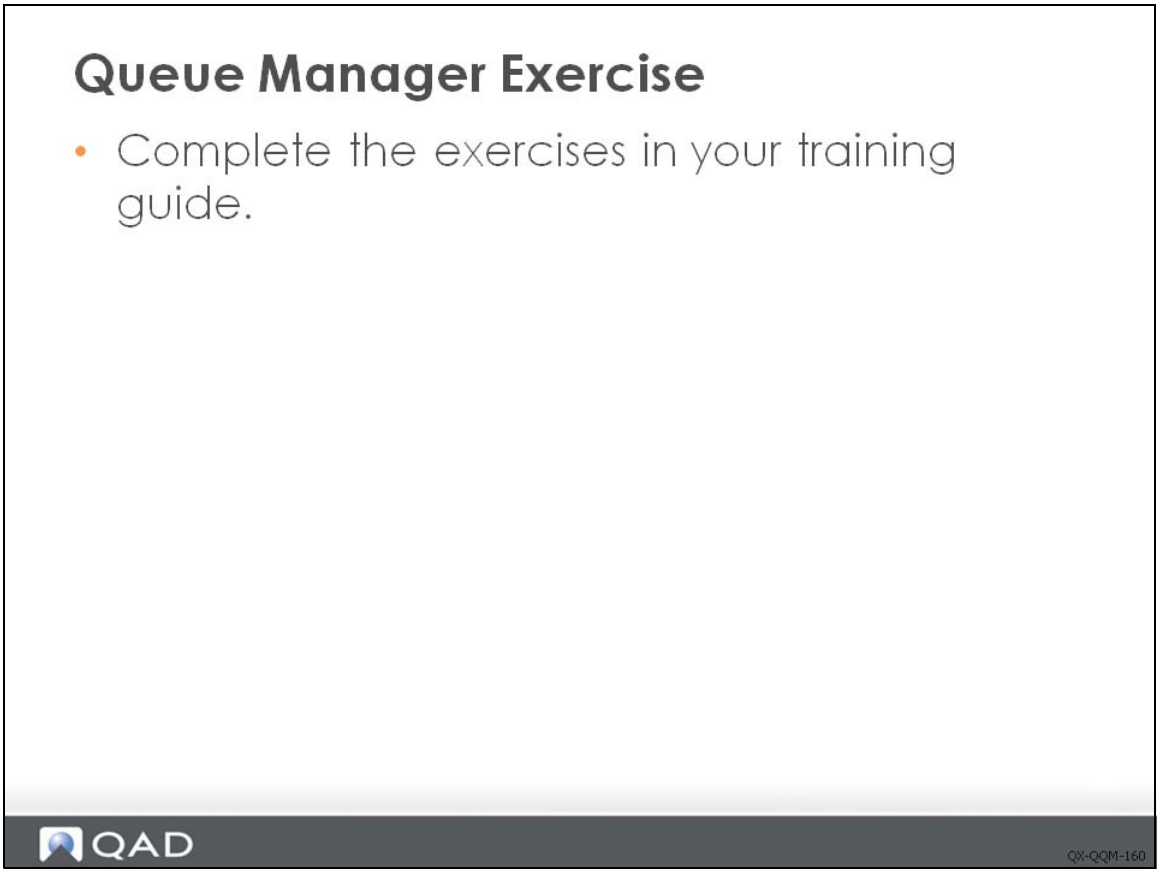
Editing/Resubmitting Requests



You can edit messages that have failed QXtend processing to correct data errors and then resubmit them for reprocessing. If no editing is required, you can just resubmit them. You cannot edit or resubmit successful/warning messages.

When editing the request make sure that the XML is not corrupted, otherwise processing will fail. The XML is not validated during the editing process or when the changes are saved.

Exercise: Queue Manager



The following list shows a number of key concepts used in the Queue Manager in QXI. In each statement below, fill in the correct term from the list.

`.ok`	`.err`
sorted, filtered, and deleted	`.req` extension
`environmentmanager.xml`	`.wrn`
XML format	initialized
Red	pending
Gray	Yellow

1. The Queue Manager is an optional interface that lets you accept QDoc request documents from external applications into a directory structure. The requests must be in _____ and be named with a _____.
2. Before using the Queue Manager, it must be _____ by editing the _____ file.
3. Once processing is complete, responses are placed in a response queue for the external application. Responses use the following extensions:
   - _____: The QDoc processed correctly.
   - _____: The QDoc processed but encountered warnings.
   - _____: The QDoc request failed.

4 After starting the Queue Manager you can view the individual queues. Documents in the queue can be _____.

5 The queue document list in the Queue Manager is color coded:

- ___ lines are documents that returned an error.
- ___ lines are messages that processed but a warning was returned.
- Green lines are messages that are either ___ or in process.
- ___ lines are successfully processed message.

Lab: Queue Manager

QAD QXtend 1.7.2 has been installed with both QXI and QXO components. In the previous labs you performed the basic configuration that is required to enable requests to be processed by QXI & QXO. In the following lab exercises you will see how to use the Queue Manager to process requests placed into an operating system directory.

`<LabHomeDirectory> = C:\QXtendTraining\Labs\07-QueueManager\`

One of the components that are part of every QXI installation is the Queue manager, which allows you to monitor a directory for messages that needs to be processed and passed to QXI. The exercises in this lab describe how to enable the Queue Manager and use it to process requests to QXI.

1. Enabling Queue Manager

The Queue Manager functionality is installed as part of the standard QXI installation. However, it is not enabled and is not available until it has been manually enabled. The process of enabling the Queue Manager requires you to manually edit one of the QAD QXtend configuration files that is held on the QXI server.

To enable the Queue Manager:

1. Open Putty by using the icon on the training Windows machine Desktop.
2. Double-click the `qaddemo` saved session.
3. Log in as user `demo-admin`, password `qad`.
4. Change directory to
   `/dr01/tomcat/8080/webapps/qxi/WEB-INF/conf`
5. Edit the `environmentmanager.xml` file:
   a. vi `environmentmanager.xml`
   b. Change the text below from:

```
<!--
<manager class="com.qad.qxtend.queue.directory.DirectoryManager"
param="/dr01/tomcat/8080/webapps/qxi/qxtendQueues"/>
-->
```

   to:

```
<manager class="com.qad.qxtend.queue.directory.DirectoryManager"
param="/dr01/tomcat/8080/webapps/qxi/qxtendQueues"/>
```

   by removing the “<!—“ at the beginning and the “-->” from the section.
6. Save your changes to this file.

1.1 Restart QXI Web Application

The changes that you made in the previous step do not take effect automatically; the QXI Web application must be stopped and started:

1 Open the Tomcat home page in Internet Explorer: `http://qaddemo:8080`

2 Select the Tomcat Manager link.

3 Enter the User Name of `admin`.

4 Enter the Password of `mfgpro`.

5 Find the entry for the QXI Web application:

Applications

Path	Display Name	Running	Sessions	Commands
/	Welcome to Tomcat	true	0	Start Stop Reload Undeploy
/QADDeployServ.V2	QAD Deployment Configuration Server	true	0	Start Stop Reload Undeploy
/epub	QAD Content Server 1.0.0.63	true	0	Start Stop Reload Undeploy
/host-manager	Tomcat Manager Application	true	0	Start Stop Reload Undeploy
/manager	Tomcat Manager Application	true	0	Start Stop Reload Undeploy
/qadhome	QAD Applications 2.9.0.88	true	0	Start Stop Reload Undeploy
/qadui	QAD Desktop 2.10.90.59	true	1	Start Stop Reload Undeploy
/qxi	QXtend Inbound 1.6.3.26	true	0	Start Stop Reload Undeploy
/qxo	QXtend Outbound 1.6.3.26	true	1	Start Stop Reload Undeploy
/search	QAD Search Engine 1.0.0.312	true	0	Start Stop Reload Undeploy
/webdav	Webdav Content Management	true	0	Start Stop Reload Undeploy

6 Select the Stop link to stop the QXI application instance.

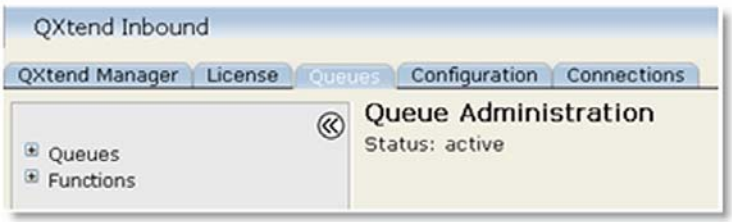
7 Once QXI has been stopped, select the Start link to restart QXI.

Applications

Path	Display Name	Running	Sessions	Commands
/	Welcome to Tomcat	true	0	Start Stop Reload Undeploy
/QADDeployServ.V2	QAD Deployment Configuration Server	true	0	Start Stop Reload Undeploy
/epub	QAD Content Server 1.0.0.63	true	0	Start Stop Reload Undeploy
/host-manager	Tomcat Manager Application	true	0	Start Stop Reload Undeploy
/manager	Tomcat Manager Application	true	0	Start Stop Reload Undeploy
/qadhome	QAD Applications 2.9.0.88	true	0	Start Stop Reload Undeploy
/qadui	QAD Desktop 2.10.90.59	true	1	Start Stop Reload Undeploy
/qxi	QXtend Inbound 1.6.3.26	false	0	Start Stop Reload Undeploy
/qxo	QXtend Outbound 1.6.3.26	true	1	Start Stop Reload Undeploy
/search	QAD Search Engine 1.0.0.312	true	0	Start Stop Reload Undeploy
/webdav	Webdav Content Management	true	0	Start Stop Reload Undeploy

8 Open `http://qaddemo:8080/qxi`

9 You will now see a Queues tab that was not available before:



The Queue Manager is now enabled.

2. Configuring Queues

Once the Queue Manager is enabled, it has a basic default configuration that includes one predefined queue. All of the queues managed by the Queue Manager share the same parent directory. The Queue Manager can manage only one queue master directory. By default the master queue directory is in `<tomcat-home>/webapps/<qxi-webapp>/qxtendQueues`. Any new queues created will be created under this directory.

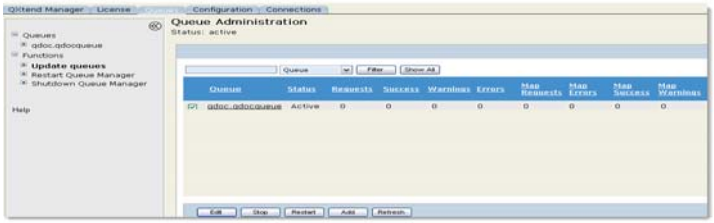
2.1 Review Existing Queue Configuration

The Queues currently managed by the Queue Manager can easily be reviewed and modified.

1. Open the QXI page in Internet Explorer: `http://qaddemo:8080/qxi`
2. Select the Queues tab.
3. Select the Queues node in the menu tree. This displays a list of the queues currently being managed. There will be one queue, the default queue called `qdocqueue`.
4. Select `qdoc.qdocqueue` from the menu tree. This shows you all requests currently in the queue; initially this will be empty.

2.2 Update Queue Configuration

1. Open the QXI page in Internet Explorer: `http://qaddemo:8080/qxi`
2. Select the Queues tab.
3. Select the Functions node in the menu tree.
4. Select Update queues from the menu.
5. Select the check box next to `qdoc.qdocqueue`.



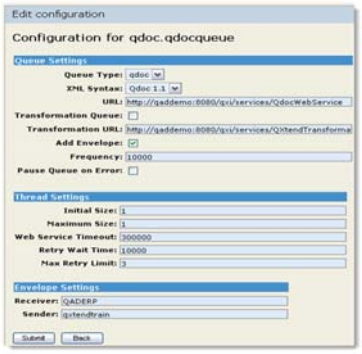
6. Click the Edit button.
7. Set the XML Syntax to QDoc 1.1.
8. Select the Add Envelope check box (this lets you post just the body to the queue and will automatically add the SOAP envelope entries before processing the request in QXI).

9 In the Envelope Settings section set the following values:

a Receiver = `QADERP`

b Sender = `qxtendtrain`

These values will be used when creating the SOAP Envelope before invoking QXI.



10 Click the Submit button to save your changes.

2.3 Create New Queue

You also can create new queues from the QXI UI. When you do this, QAD QXtend creates the required directory structure under the directory that is being managed by the Queue Manager.

1 Open the QXI page in Internet Explorer: `http://qaddemo:8080/qxi`

2 Select the Queues tab.

3 Select the Functions node in the menu tree.

4 Select Update queues from the menu.

5 Click the Add button.

6 Enter the following details:

a Queue Name = `Training`

b Queue Type = `qdoc`

c URL = `http://qaddemo:8080/qxi/services/QdocWebService`

7 Click the Create button.

8 Set XML Syntax = QDoc 1.1.

9 Click the Submit button to commit changes.

10 Select the Queues tab.

11 Select the Functions menu.

12 Select Update queues from the menu.

You now see the new Training queue listed.

3. Process Transactions Using the Queue

The Queue Manager processes requests that are placed in the queues request directory. For a file to be processed, it must have a `.req` extension. If the request file does not have a `.req` extension, it will be ignored by the Queue Manager.

The Queue directories managed have the following structure:

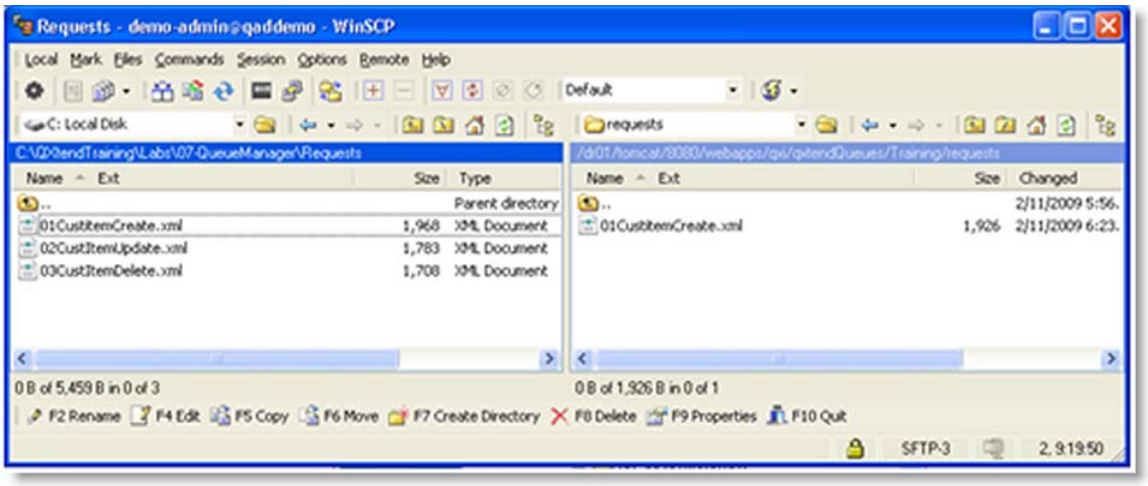
- `<QueueMgrDirectory>/<queuename>/requests` - contains the requests that are waiting to be processedby the Queue Manager. This is where request files must be placed with the `.req` extension.
- `<QueueMgrDirectory>/<queuename>/responses` - contains request that have been processed. The request file is copied here after it has been processed. The response message from the processing is also written here.
- `<QueueMgrDirectory>/<queuename>/system_failures` - contains requests that were being processed when a system failure occurred. These must be reprocessed.

3.1 Process Successful Request Messages

In this first exercise we will process messages that are formatted as a valid QDoc, including a SOAP envelope.

3.1.1 Create Customer Item

1. Open the Win SCP application that is installed on the Windows image.
2. Open the stored session `demo-admin@qaddemo`.
3. Navigate to the Training queue requests directory on the server:
   `/dr01/tomcat/8080/webapps/qxi/qxtendQueues/Training/requests`
4. Navigate to the location of the sample requests on the Windows client:
   `C:\QXtendTraining\Labs\07-QueueManager\Requests`
5. The above lab directory contains sample requests you will process using the queue in the following steps. The requests have a `.xml` extension so that when they are copied into the requests directory, they are not picked up until you change the file extension to `.req`.
6. Copy `01CustItemCreate.xml` from the client to the requests directory.



7 Change the permissions of the file:

   a Right-click the file.

   b Click Properties.

   c Select all of the Write boxes.

8 Using WinSCP, rename `01CustItemCreate.xml` to `01CustItemCreate.req`.

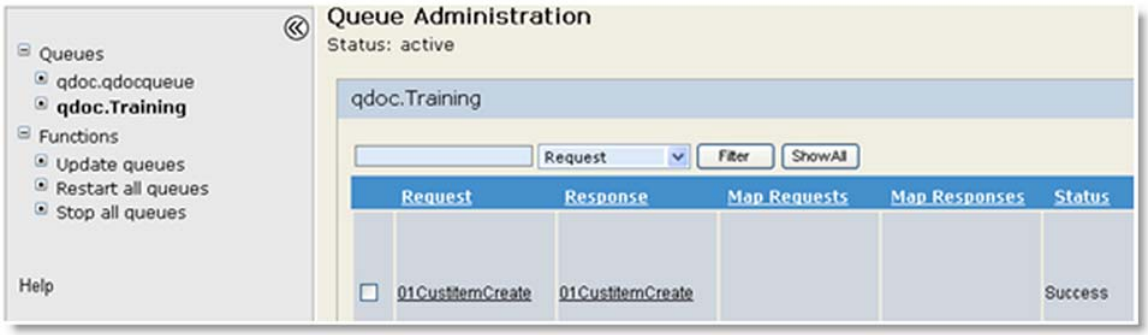
Check the Queue

1 Open the QXI page in Internet Explorer: `http://qaddemo:8080/qxi`.

2 Select the Queues tab.

3 Expand the Queues node in the menu tree.

4 Select the qdoc.Training menu. After a time you will see a successful request.



5 View the request and response details by clicking the link on the request/response name.

6 Check the QAD EE application to make sure that the data has been loaded.

7 Repeat the process above for the update and delete QDocs.

3.2 Process Error Request Message

1 Open the Win SCP application that is installed on the Windows image.

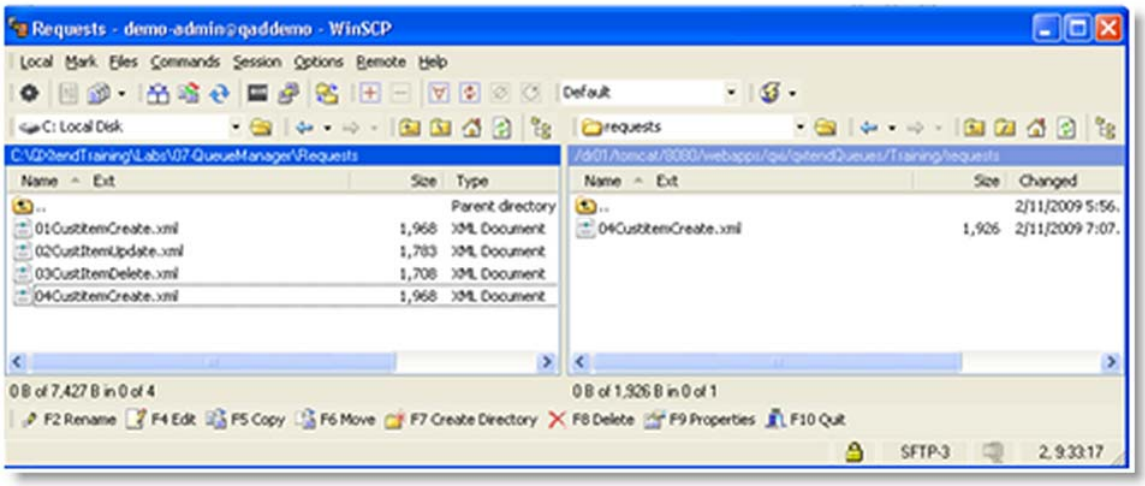
2 Open the stored session `demo-admin@qaddemo`.

3 Navigate to the location of the Training queue requests directory on the server:
`/dr01/tomcat/8080/webapps/qxi/qxtendQueues/Training/requests`

4 Navigate to the location of the sample requests on the windows client:
`C:\QXtendTraining\Labs\07-QueueManager\Requests`

5 The above lab directory contains sample request that you are going to process using the queue in the following steps. The requests have a `.xml` extension so that when they get copied into the requests directory, they are not picked up until you change the file extension to `.req`.

6 Copy `04CustItemCreate.xml` from the client to the requests directory.



7 Change the permissions of the file:

- a Right-click the file.
- b Click Properties.
- c Select all of the Write boxes.

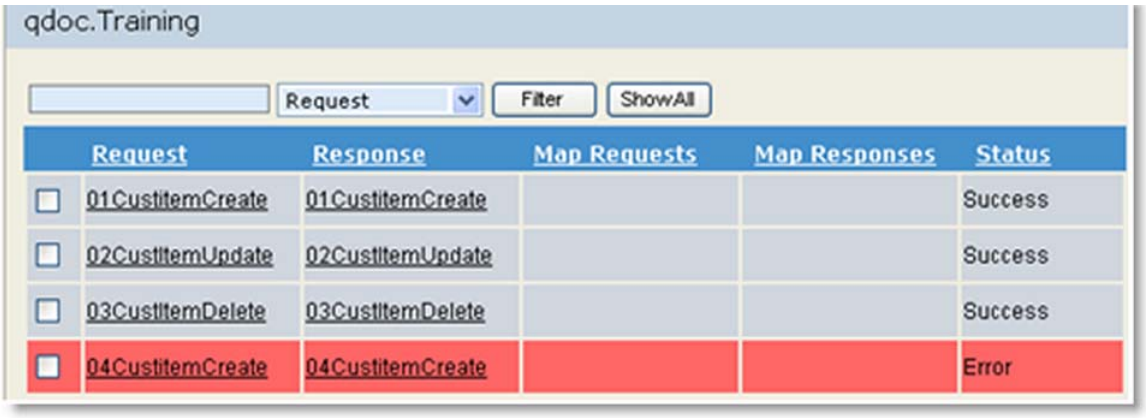
8 Using WinSCP, rename the `04CustItemCreate.xml` file to `04CustItemCreate.req`.

Check the Queue

1 Open the QXI page in Internet Explorer: `http://qaddemo:8080/qxi`

2 Select the Queues tab.

3 Select the qdoc.Training menu. An error request status displays:



4 The QDoc failed because the customer number was invalid. Open the response, look at the details, and locate the exception information.

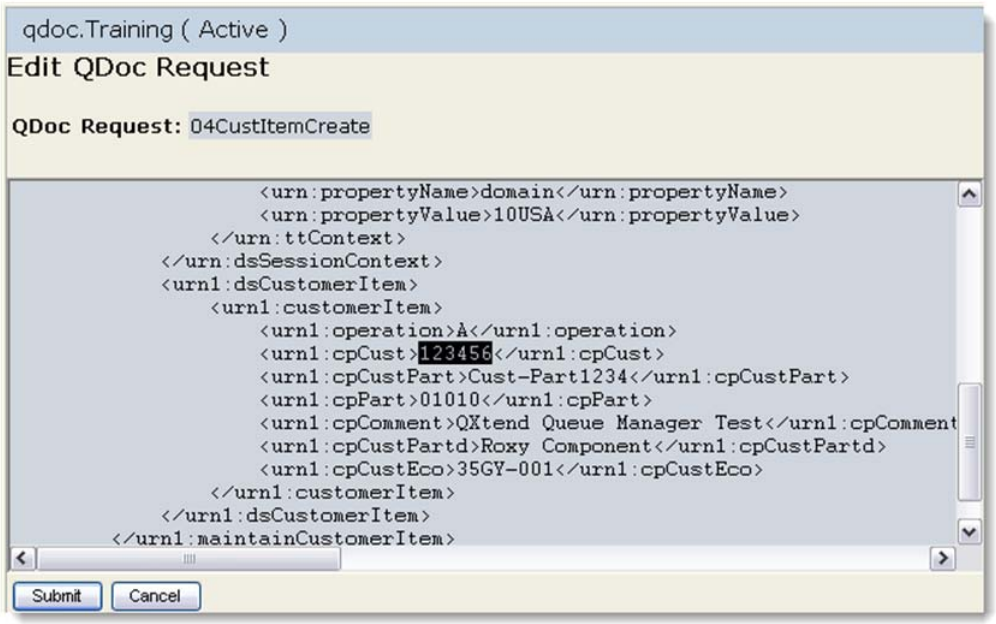
a Click the response file name.

b Click the View Detail button and review the response XML.

c Correct the message and change the `cpCust` node to `10C1000`.

d Click the Edit Request button.

- Change the `cpCust` node to `10C1000`.
- Click the Submit button.



The message will process successfully.

3.3 Automatically Add SOAP Envelope to Request Message

The qdoc.qdocqueue has been configured to automatically add a SOAP Envelope to any request that is placed in the queues request directory. Review the `<LabHomeDirectory>/request/05CustItemCreate.xml` file; it does not have a SOAP envelope.

You will now process this message using the qdocqueue.

1. Open the Win SCP application that is installed on the Windows image.
2. Open the stored session `demo-admin@qaddemo`.
3. Navigate to the location of the Training queue requests directory on the server:
   `/dr01/tomcat/8080/webapps/qxi/qxtendQueues/qdocqueue/requests`
4. Navigate to the location of the sample requests on the windows client:
   `C:\QXtendTraining\Labs\07-QueueManager\Requests`
5. The above lab directory contains sample request that you will process using the queue, The requests have a `.xml` extension so that when they are copied into the requests directory, they do are not picked up until you change the file extension to `.req`.
6. Copy the `05CustItemCreate.xml` from the client to the requests directory.
7. Change the permissions of the file:
   a. Right-click the file.
   b. Click Properties.
   c. Select all of the Write boxes.
8. Using WinSCP rename the `05CustItemCreate.xml` file to `05CustItemCreate.req`.

Check the Queue

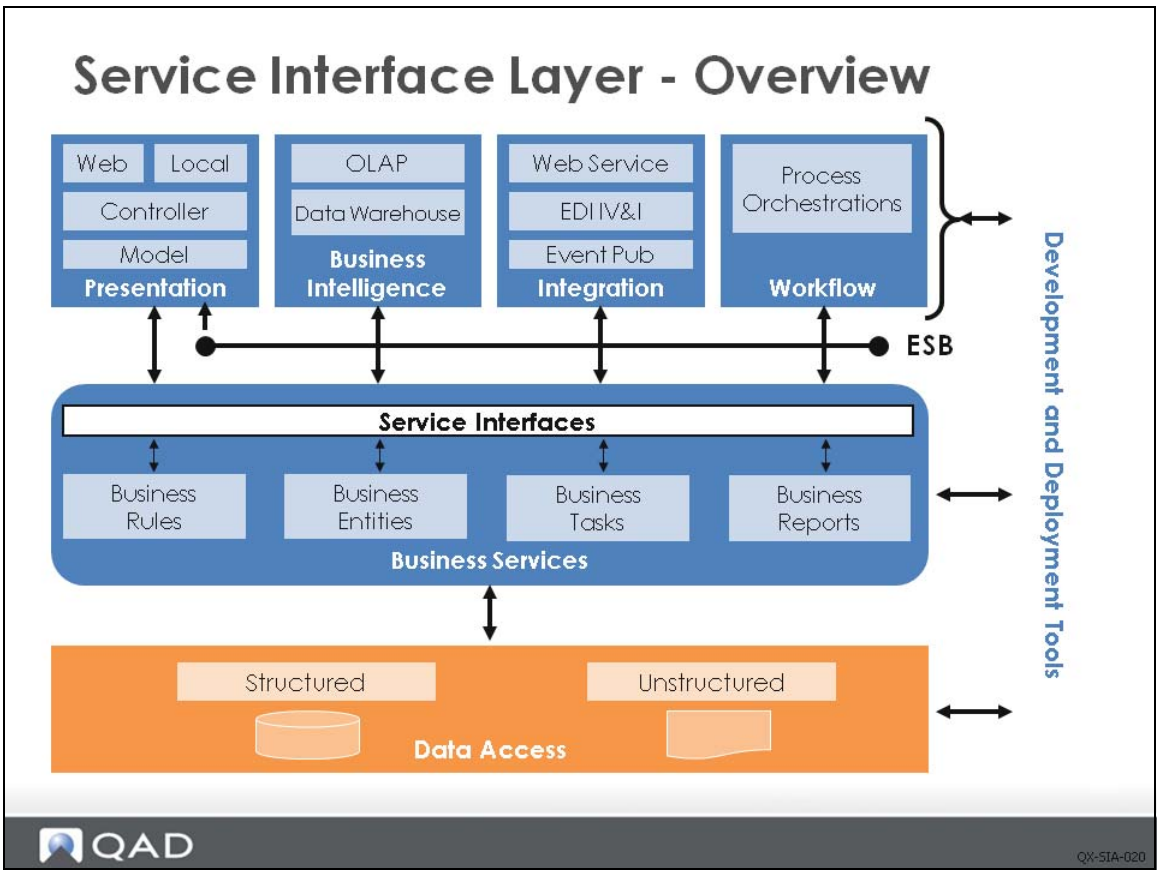
1. Open the QXI page in Internet Explorer: `http://qaddemo:8080/qxi`
2. Select the Queues tab.
3. Select the qdoc.Training menu.

After a time you will see a success request status.

Chapter 8

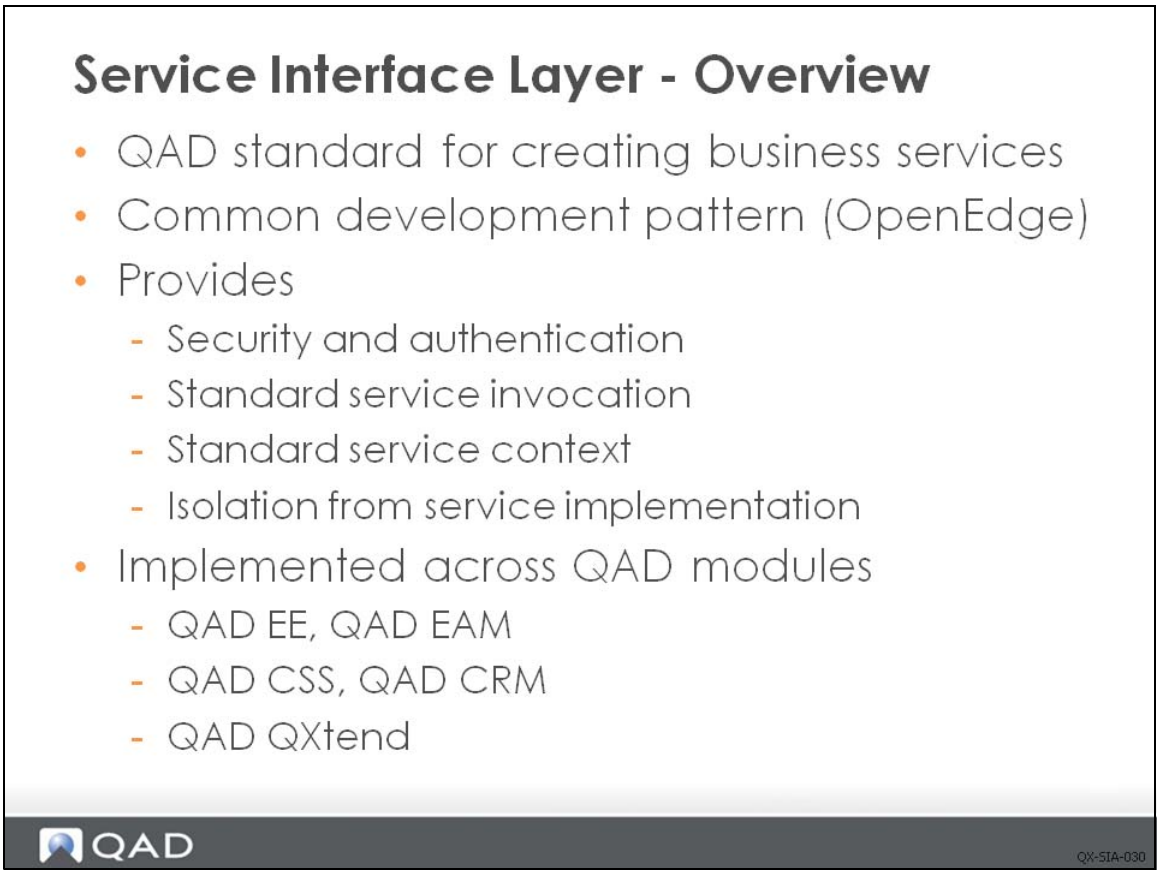
Service Interface Layer

Overview



The service interface (SI) layer is a fundamental architectural concept developed at QAD. This layer is a set of common infrastructure components and a set of development standards that can be used to develop business services for any OpenEdge-based QAD application.

The service interface enables any QAD application to follow a common pattern to provide the services that are critical when developing applications that follow service oriented architecture (SOA) design principles. The common service interface enables other components such as integration platforms, user interface toolsets, or reporting tools to access your services in the same way without having to understand the underlying implementation of the service.



The service interface layer provides infrastructure and standards used in QAD application development to ensure that any external business services (APIs) follow a common pattern to expose those services. Previously, QAD application development teams have taken different approaches when developing APIs, resulting in inconsistent APIs and different interoperability tools and standards. The service interface ensures that OpenEdge development teams use common standards to develop APIs, creating a single set of interoperability tools and services.

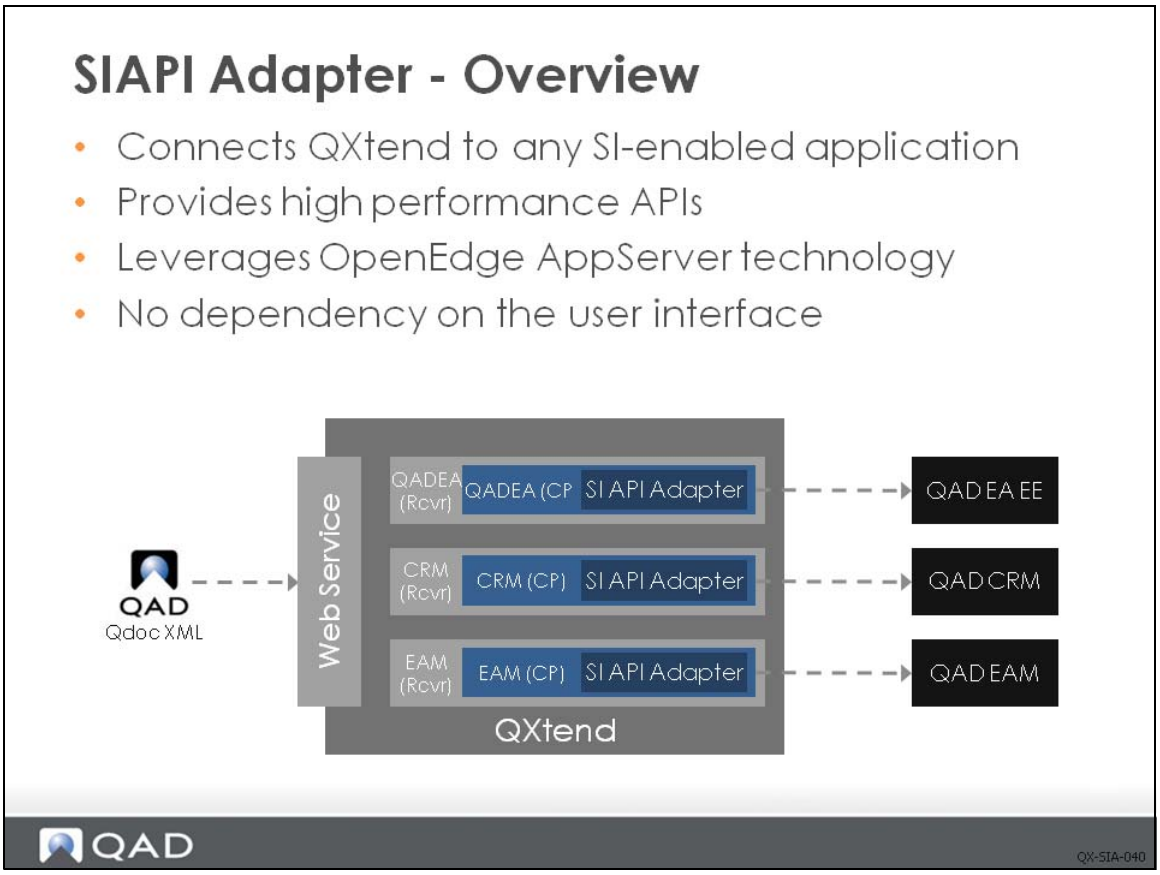
Key features handled by the service interface include:

- Security and authentication: These services are common to any application and — of course — different applications implement these services differently. The service interface defines a common interface that is implemented by any application that uses it for security and authentication-related actions. The service interface layer does not provide the implementation—that is the job of the application team. The application implements the interface and connects the service interface to its security and authentication services.
- Service invocation: A common API invocation layer is provided that is responsible for invoking the application service for external components. The generic service invocation requires that the API called from the service interface follows a set standard and interface. The advantage of having a common invocation component is that external applications only have to be able to invoke services through the service interface, which means that one component can call any service implemented using the service interface.
- Service context: Requests to any service need to carry context information—such as user, domain, and other application-specific information—to determine how to process the request. The service interface defines a standard for this and provides methods to access request

context information. Service context is critical so that we can work with stateless and state-free AppServers, and therefore scale the AppServer pool. In effect, we manage our session state, rather than having OpenEdge do it for us.

- Isolation: The overall goal of the service interface is to provide common components that isolate the caller from the implementation details of the target application/API.

The service interface is already used or under development with several QAD Enterprise Applications. The Enterprise Edition Financials has been developed with a complete SOA infrastructure, and provides access to the Financial APIs via the SI. QXtend also has APIs implement the SI; products such as EAM, CRM, DOM, and CSS all implement the SI. The number of applications that implement the SI will continue to increase.

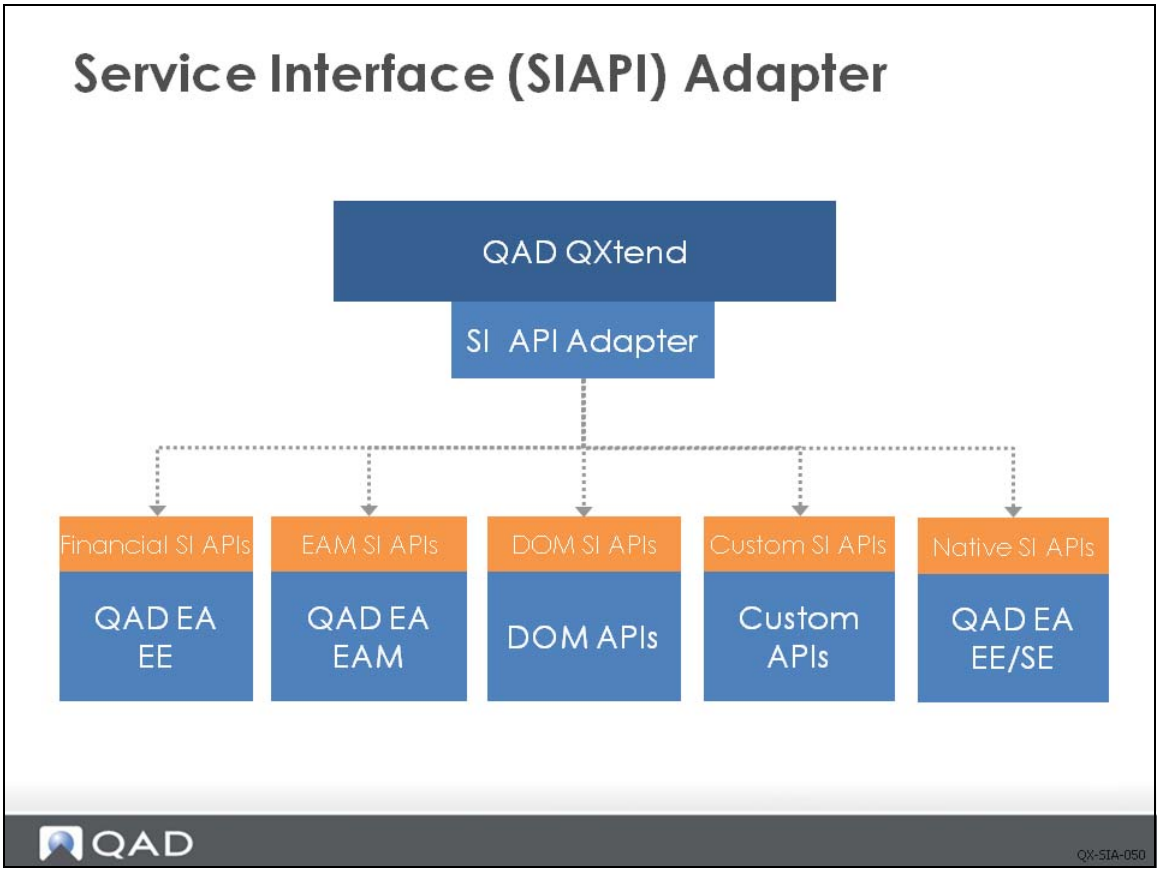


The service interface (SIAPI) adapter in QXI leverages the standard service invocation features of the SI layer to provide an adapter that can process requests to any business application service that has been implemented using the SI layer. The advantage of using the SIAPI adapter is that no additional QXtend development work is required to support new SI-enabled applications.

The APIs that are provided via the SIAPI differ fundamentally from the UIAPI adapter: the APIs are pure business logic and do not depend on the UI. Executing business logic without the UI greatly improves the performance of the API and enables many more messages to be processed.

The configuration of the SIAPI adapter connection pool also differs from the UI API. The primary difference is that messages are processed using an OpenEdge AppServer instead of data being passed through a telnet connection. Configuring the connection pool requires you to enter details about the AppServer instance that is going to execute requests.

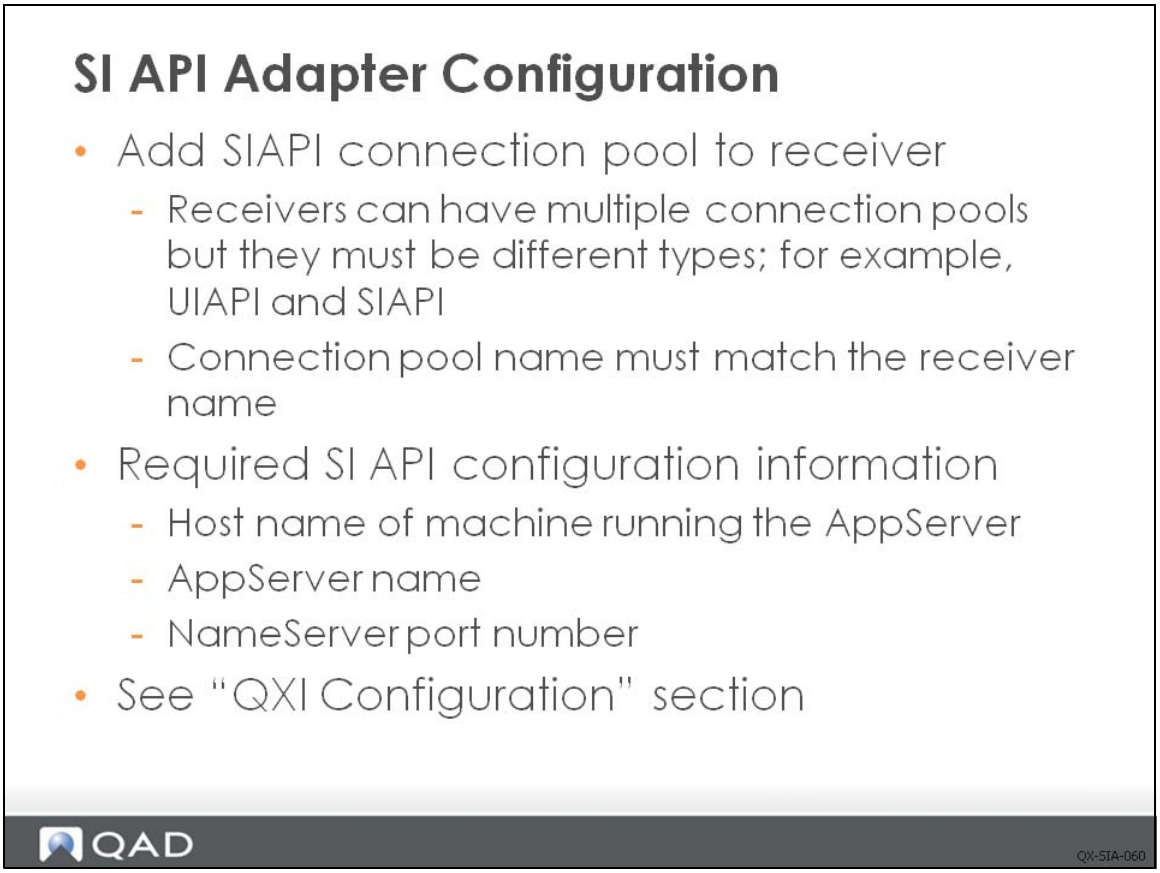
Service Interface Adapter



The SIAPI adapter allows you to add new SI-enabled applications to QXtend easily. Four or five applications either already use the SI or will do soon; this number will increase. As these QAD applications appear, their APIs will be accessible from QXI, allowing a common application-to-application integration methodology across all QAD products.

Customers and services can develop custom APIs that implement the SI; those custom APIs will be available through QXtend, and the same applies for QAD application partners.

Configuring the SIAPI



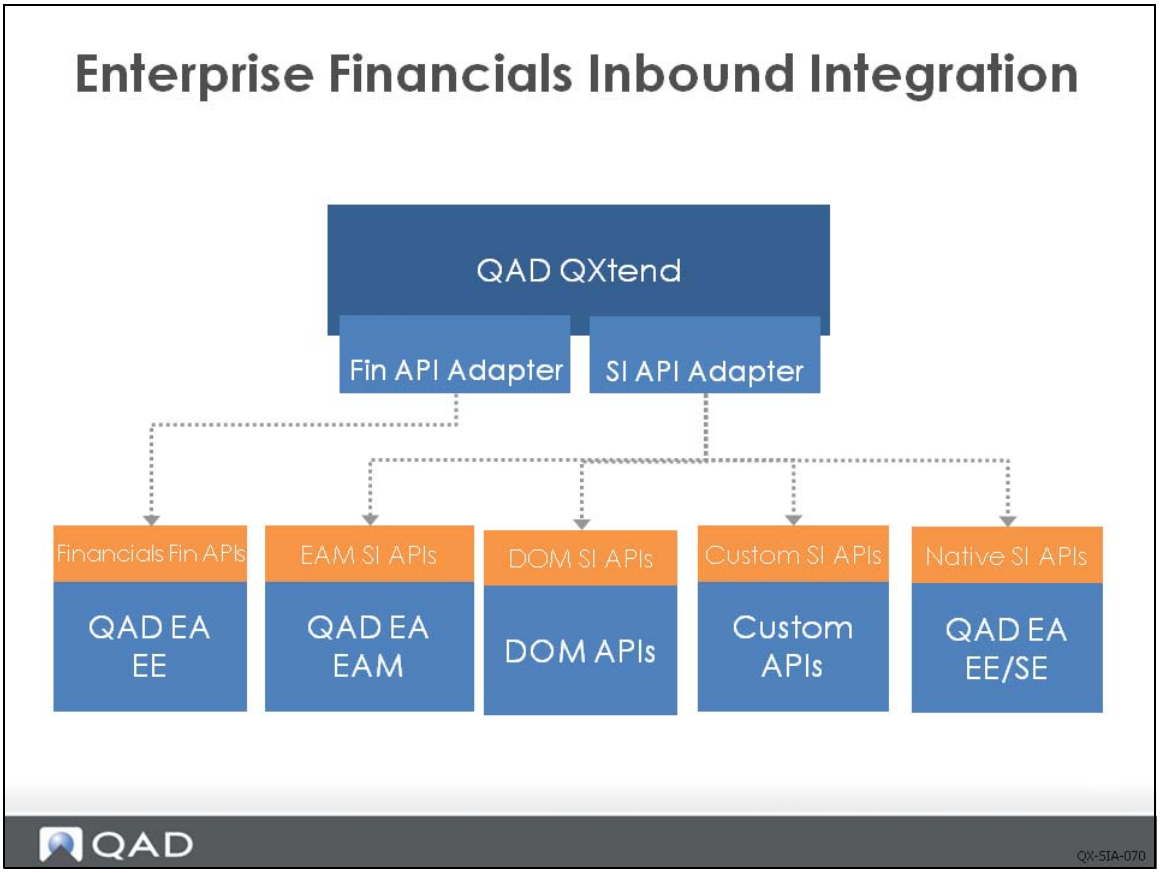
Configuring the SIAPI adapter is described Chapter 2, “QXI Configuration,” on page 51. An SIAPI pool can be added to any receiver set up within QXI. However, receivers can only be connected to one connection pool for each adapter type—two SIAPI connection pools cannot be added to a receiver. A common configuration for QAD EA EE is for a receiver to have a UI API connection pool for the operational APIs, and an SIAPI connection pool for the Financial APIs. When creating a second connection pool for a receiver, the connection pool name must be identical to the name of the receiver.

When you configure a SI API connection pool, you need the following information to connect to the AppServer:

- Host: Name or IP address of the machine that is running the AppServer instance you want to connect to.
- AppServer: Name of the AppServer configured to accept service interface requests.
- Port: Port number of the NameServer that the AppServer is registered with.

For details on creating an SI API connection pool, see “Connection Pools” on page 64.

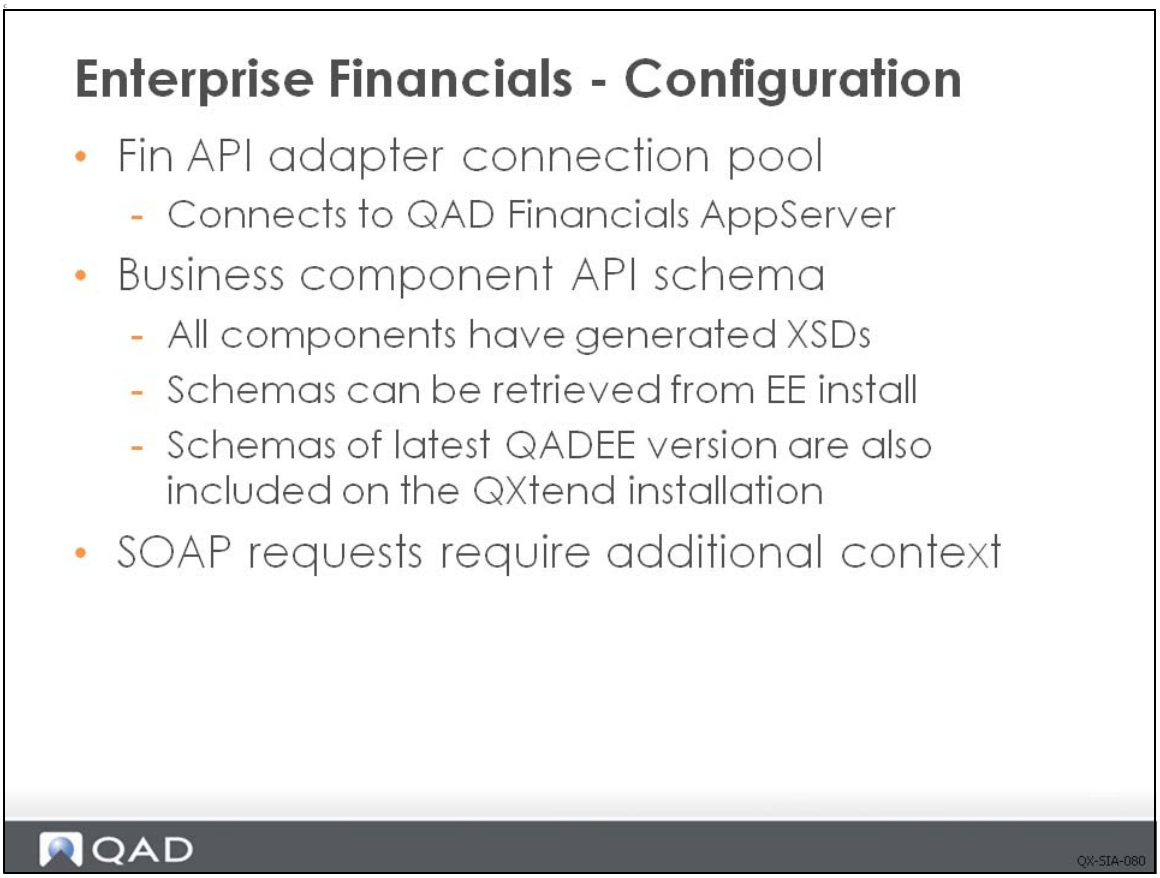
Enterprise Financials Inbound Integration



The Enterprise Financials module in QAD EE now follows the latest development standards. The result is an SOA-enabled application and several services that have no dependence on the UI. The service interface is also supported by the new Financials, meaning that you can use the services available on the Financials business components with QXtend.

The Fin API is one kind of SI API but for QAD Enterprise Financials. For QAD Enterprise Application, the Fin API adaptor calls Financials AppServer (only available in QADEE) and SI API adaptor calls Native API AppServer (available in both QADSE and QADEE). The main reason we use the name Fin API is that for each receiver, only one connection pool can be defined for each type of adaptor.

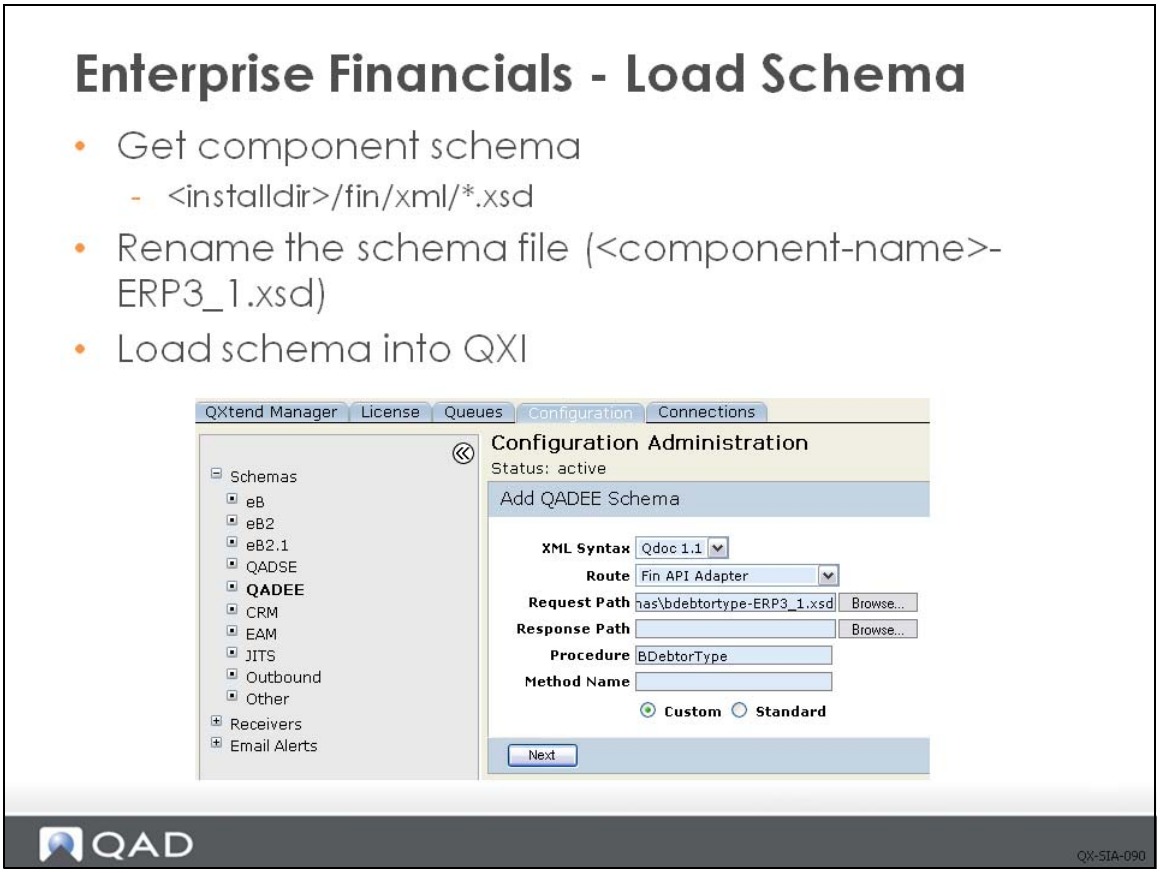
Configuring SIAPI in Enterprise Financials



QAD QXtend leverages the SI support of the new Enterprise Financials to access to the Financials APIs through the Fin API adapter. A Fin API connection pool must be created that connects to the Financials AppServer.

Currently QXtend includes most of latest schemas for Financials but you can retrieve the schema from EE install as well. For example, QXtend 1.7.2 was released together with QAD 2011 EE so it includes schemas for QAD 2011 EE Financials components. However, if you want to use QXtend 1.7.2 with QAD 2010 EE, please retrieve the schemas from QAD 2010 EE install.

Loading a Schema into Enterprise Financials



To use one of the supported Financials business components with QXtend, you must manually retrieve the schema and load it into QXtend by doing the following:

1 Get schema. The schema files for the business components are located in the Financials instance in the directory `<qadee-installdir>/fin/xml`. The schema naming convention is `<component-name>.xsd`. Locate the schema for the component you need and copy it locally.

2 Rename schema file. QAD QXtend requires the API schema to follow a different naming convention: `<api>-<version>.xsd`. The schema file copied from the Financial installation must be changed to `<component-name>-ERP3_1.xsd`.

3 Load schema. Load the schema file for the component into QAD QXtend using the Schemas section of QXI. Add a new schema to QADEE. When you add the schema, you need to supply the following values:

XML Syntax. For all EE financial APIs Qdoc 1.1 syntax must be used.

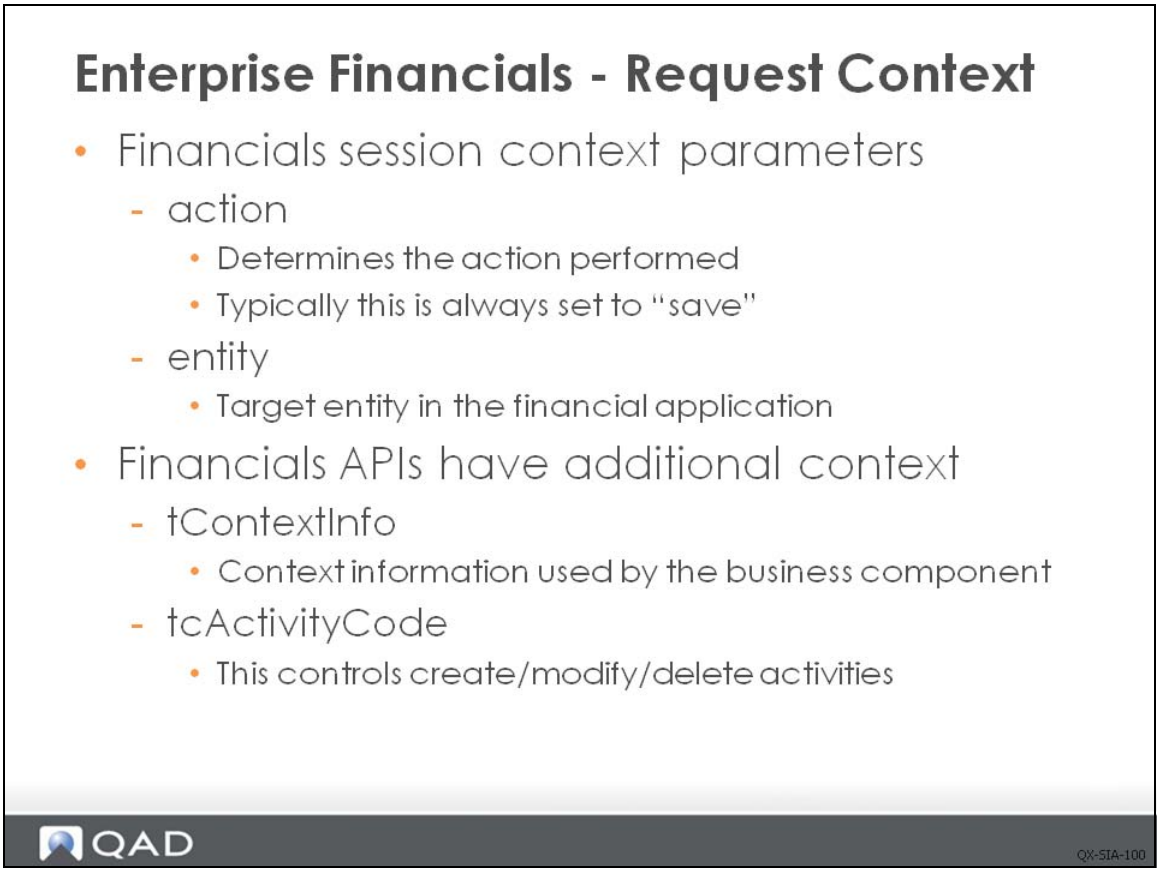
Route. All EE financial APIs are accessed through the Service Interface, choose "`Fin API Adapter`" for the route.

Request Path. The path to the renamed XML schema. The path must be accessible from the local machine.

Response Path. The path to the response file for the API. Typically Financial APIs do not have response schemas so this can be left blank. However, future releases of QAD EE may provide response schemas for the APIs, and these should be loaded if provided.

Procedure. Used by the service interface to determine the business component to load data into. Enter the name of the business component; for example, BDebtorType.

Enterprise Financials - Request Context



Requests to any of the Financials APIs require additional context information. Context information controls how the request is processed by the API. The additional context information has two types. The first type is passed as an SI session context entry in the `ttContext` table; the other type is passed in the request business component data as part of the `tContextInfo`. These are new parameters that are only required for requests being processed by a Financials API.

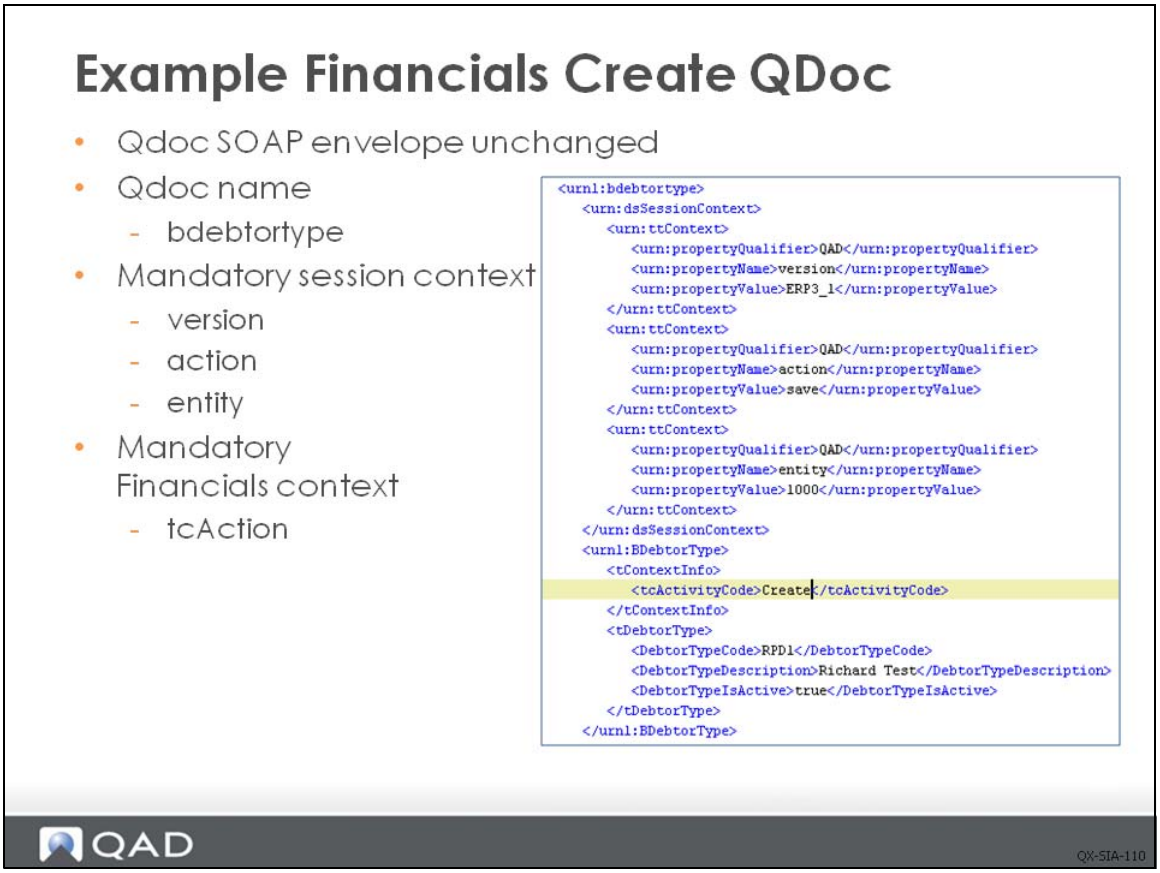
SI Session Context (ttContext)

- Action: Tells the SI that the action must be performed against the business component once the transaction has been processed. If the API being used is maintaining data, the save action must be specified to ensure that any changes processed by the API are committed.
- Entity: Defines the entity that the data in the request will be loaded into. This parameter is similar to the domain parameter for other QAD EA requests. The service interface will switch to the specified entity before processing the request.

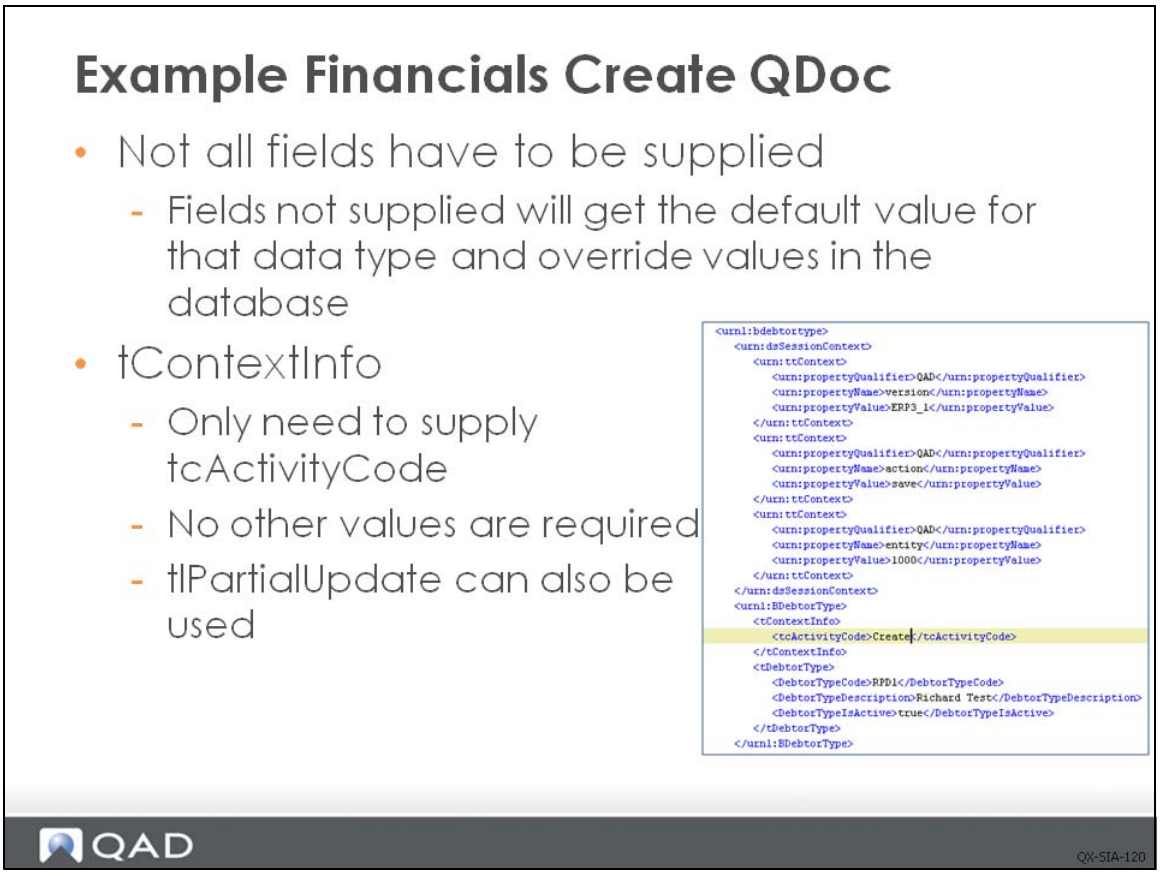
Business Component Context Information (tContextInfo)

- `tcActivityCode`: Every request to a Financials business component must include context information. When processing an API request, the `tcActivityCode` is the most important as it defines the activity being performed against the business component. The standard maintenance APIs use one of the following values: create, modify, or delete. This has the same function as the operation node in non-Financials API QDocs.

Examples



The above example QDoc message shows a create QDoc for the BDebtorType business component. The QDoc SOAP envelope is not changed when calling a Financials API. The QDoc name is always the name of the Financials component being updated. In the example this is the `<urn1:bdebtortype>` node.

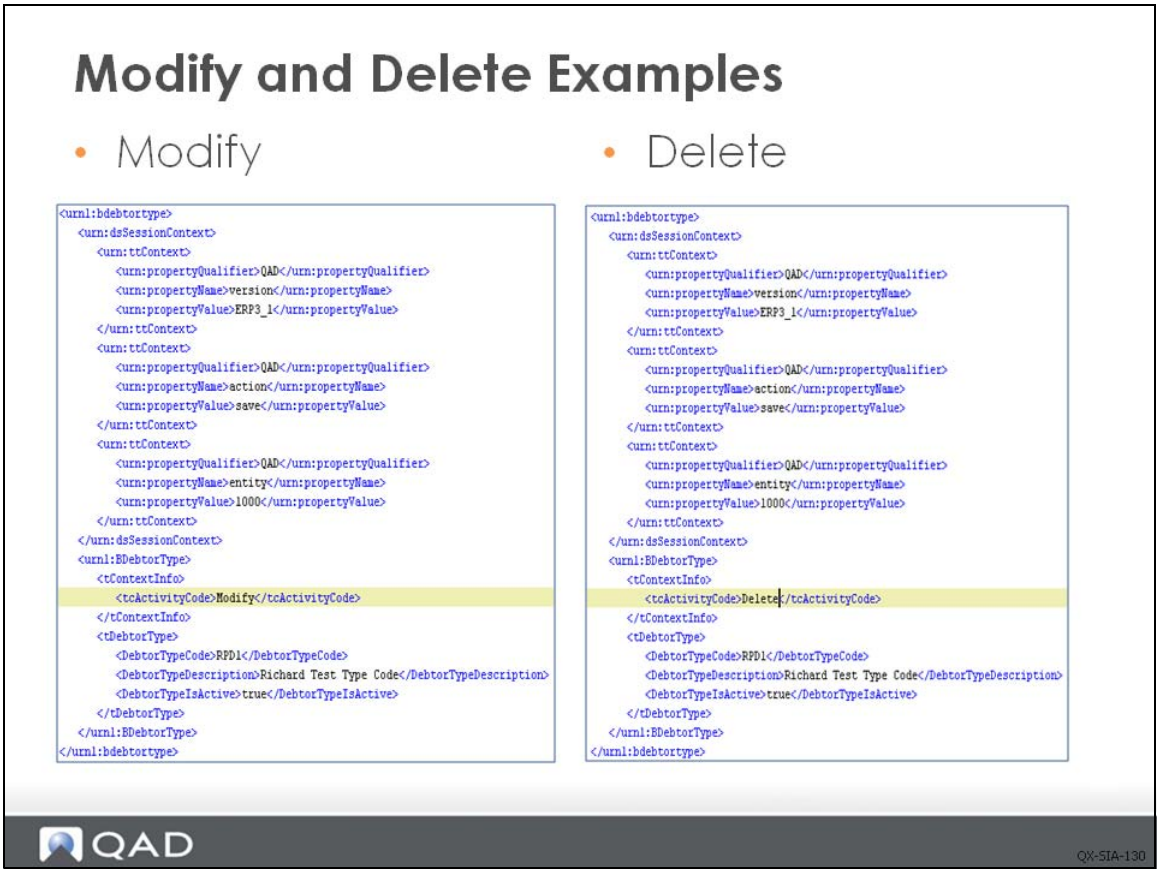


The Financial business components have many fields, but not all of them are required when using the QXtend API. Fields that have default values in the application can be omitted from a create request and will be assigned the default value for the API. Some other fields also can be omitted:

- *_ID fields: Implementing the Financials business components uses ID values to uniquely identify records within a table. These values are then used for primary keys, foreign keys, and parent-child relationships. However, ID fields provide internal information that is not intended for consumption by external applications. Instead, business keys (customer number, invoice number, and so on) are used. When calling a Financials API from QXtend, ID fields can be omitted from the request and the business keys provided instead; this helps the external system to create the request. Essentially the ID fields are not used by the QXtend API.
- Custom* fields: The custom fields store field values that are added to the Financials application through the customization framework. Only fields used for customizations need to be provided.
- LastModified* fields: These fields are updated automatically by the Financials API and are not required as part of the request.

`tlPartialUpdate` is used in update. If `tlPartialUpdate` is set to true, then during the update, system will keep the original value for fields that don not have a value supplied in request. By default `tlPartialUpdate` is set to true.

Modify and Delete Examples



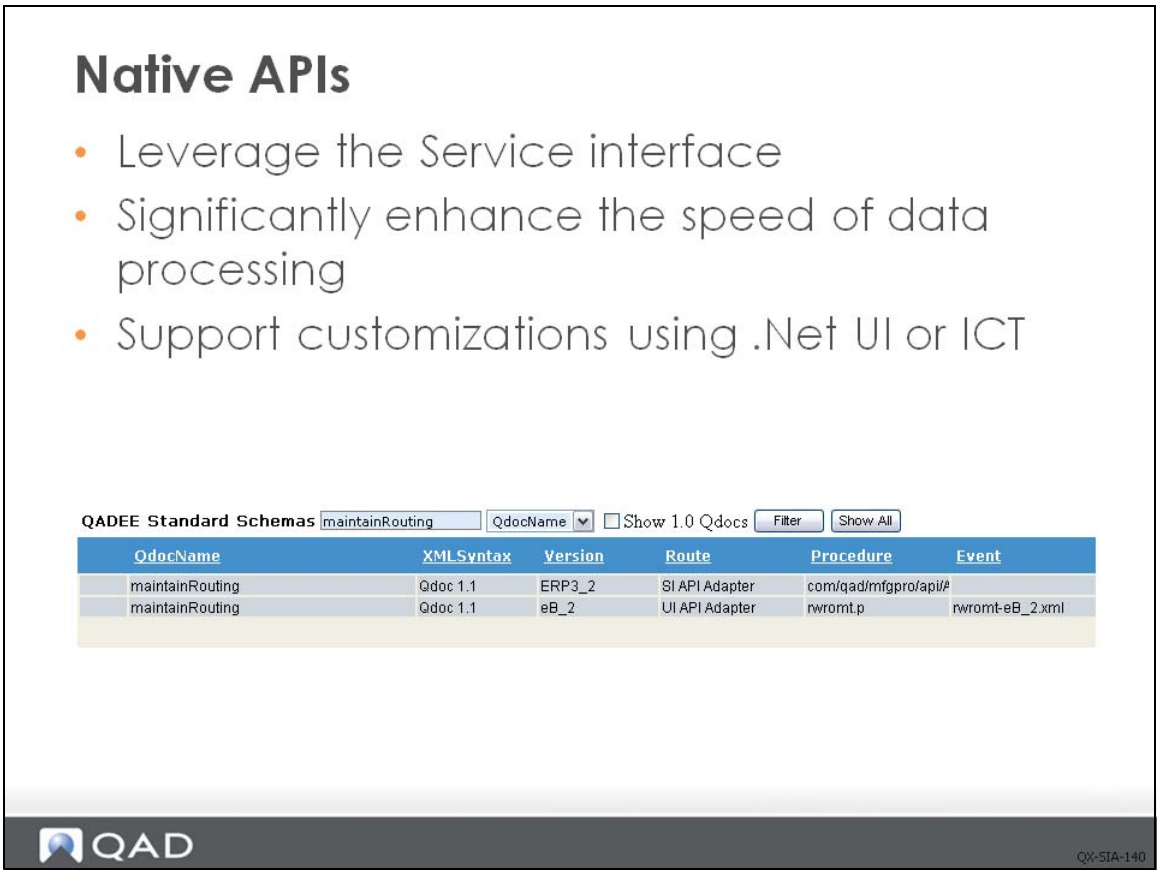
Performing a modification requires a similar request message as the create transaction but `tcActivityCode` must be set to Modify. If processing a delete, set it to “Delete”.

Empty nodes in the Financial API behave differently according to whether a “Create” action or a Modify action is being performed—it is crucial that you understand the difference:

- In a create message, omitted nodes are assigned a default value.
- In a modify message the value in the application is overwritten with a blank value.

Therefore, when processing an update make sure that all nodes are included in the request QDoc to prevent application data from being overwritten mistakenly.

Note This issue will be addressed in a future release of QAD Enterprise Edition.



The UI API adapter is the primary method used by QXI to load data into QAD applications. However, UI API adapter performance may be unacceptable when processing high volumes of data or real-time transactions.

You can use native APIs to leverage the SI interface and significantly enhance the speed of data processing. Native APIs provide a set of development patterns that can be used to wrapper existing character screens and execute the underlying business logic independent of the user interface, resulting in faster processing. Bulk loads can be performed faster, which is particularly useful when processing large business objects such as Item and Product Structure.

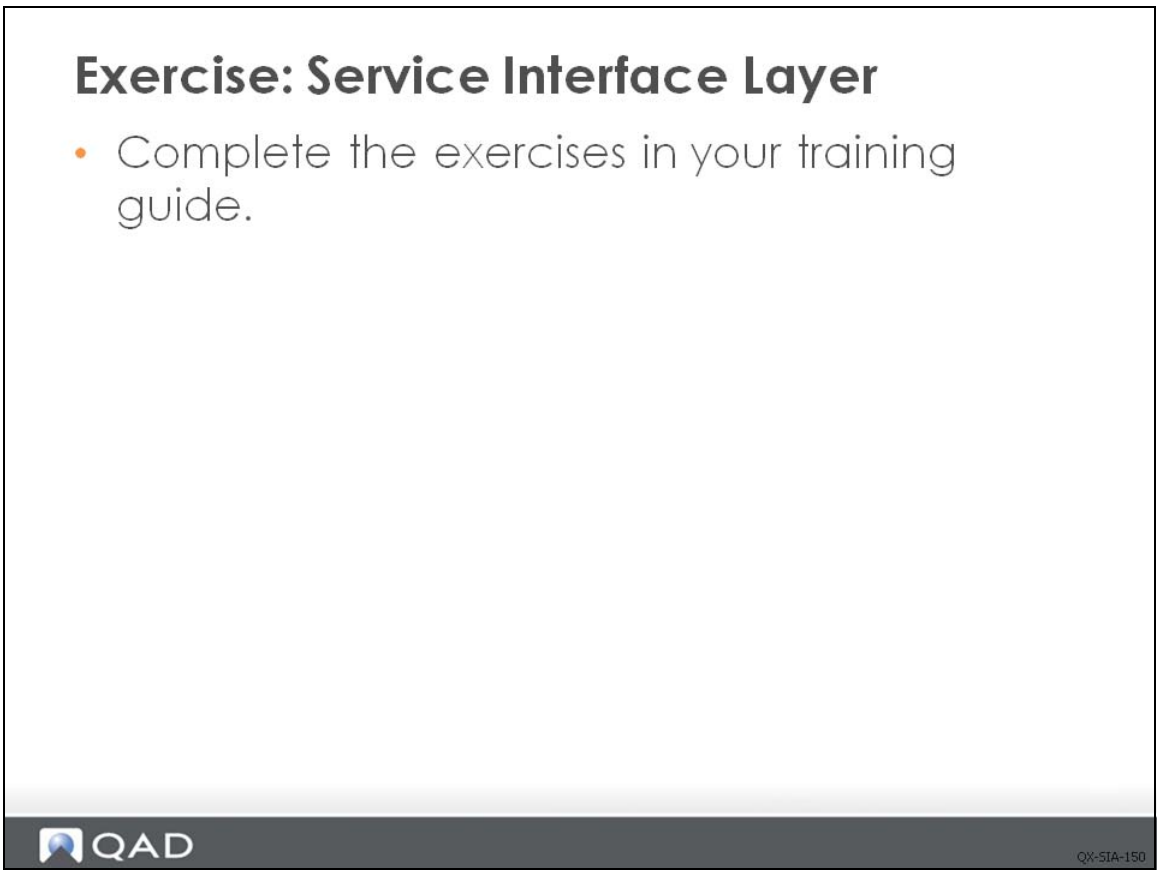
To use Native APIs, you need to set up the SIAPI connection pool and connect to Native API AppServer. Also, you need to use the APIs with Route “SI API Adaptor”. A Native API can have the same name as UIAPI but different version number.

Native APIs are available in the QAD QXtend 1.6 version; they are not available in earlier versions.

The Native API framework now supports customizations using either the .NET UI or the Integration Customization Toolkit (ICT). The framework supports features such as default values, disabled fields, custom validations, additional fields and frames, shadow tables, and custom programs.

For details on Native APIs, see Technical Reference: *QAD QXtend*.

Exercise: Service Interface Layer



The following list shows a number of key terms and concepts for the service interface layer and service interface adapter. In each statement below, fill in the correct term from the list.

OpenEdge AppServer	host
port	business logic
blank	default value
service oriented architecture (SOA)	AppServer
telnet connection	SI API adaptor
Fin API adaptor	

1. The SI enables the development of applications that follow _____ design principles.
2. APIs provided via the SIAPI are pure _____ and do not depend on the UI.
3. In the UI API, messages are processed via a _____. In the SI API, messages are processed using an _____.
4. When configuring an SI API connection pool, you need to provide the name of the _____, the _____, and the _____ number.
5. In a create message, omitted nodes are assigned a ____. In a modify message the value in the application is overwritten with a _____ value.

6 For QAD Enterprise Application, the _____ calls Financials AppServer (only available in QADEE) and _____ calls Native API AppServer (available in both QADSE and QADEE).

Lab: Service Interface Layer

QAD QXtend 1.7.2 has been installed with both QXI and QXO components. In the previous labs you performed the basic configuration that is required to enable requests to be processed by QXI and QXO. In the following lab exercises you will see how to use APIs that are implemented with the service interface. For our exercises you will use some of the QADEE Financials APIs.

<LabHomeDirectory> = C:\QXtendTraining\Labs\08-ServiceInterface\

1. Fin API

1.1 Create Fin API Connection Pool

So far you have exclusively used the UI API adapter to process requests. The UI API adapter connection pool establishes a telnet connection and cannot be used to execute service interface APIs. Instead you need to create a new SI or Fin API connection pool. The new connection pool can be assigned to the same receiver as it is a different type.

1.1.1 Create New Connection Pool

Create an Fin API connection pool for the QADERP receiver:

1. Open the QXI Web application in Internet Explorer: `http://qaddemo:8080/qxi`
2. Select the Connections tab.
3. Expand the Add Connection Pool menu.
4. Select the Add FinAPI Pool option.
5. Set the following values:

   Pool Name = `QADERP` (must be identical to receiver name created in the previous step)

   App Server Name = `qadfinlive`

   Host = `qaddemo`

   Port (NameServer): `5162`

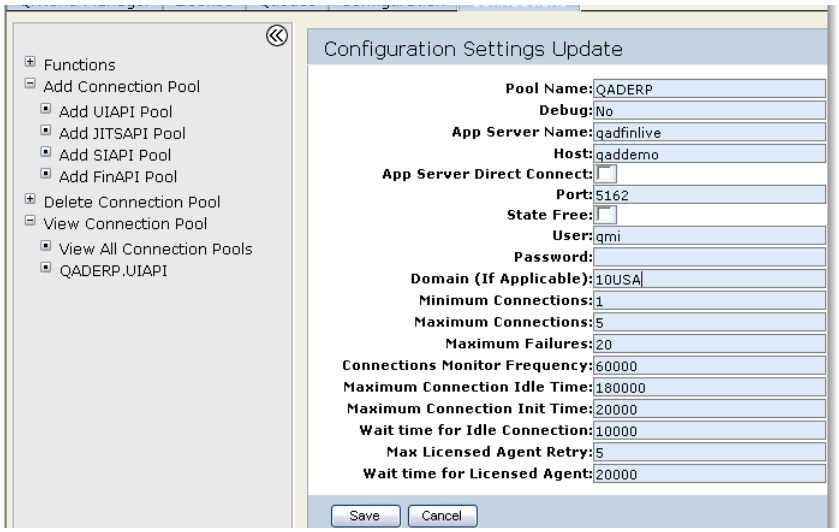
   User = `qmi`

   Password = “” (blank)

   Domain = `10USA` (this is only the default domain, the domain can be passed with the request and will override this value on when processing the request).

   Minimum Connections = `1`

   Maximum Connections = `2`
6. Click the Save button to create the new connection pool.

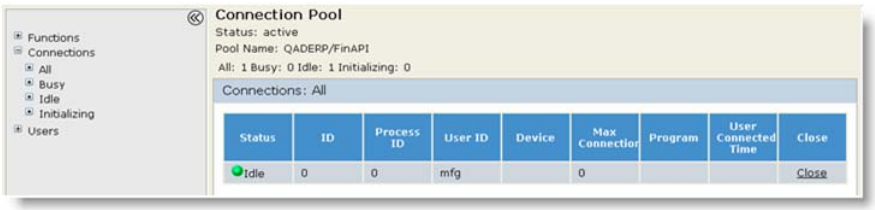


1.1.2 Validate New Connection Pool

Now you need to check your configuration is correct and that you have idle sessions available.

1. Open the QXI Web application in Internet Explorer: `http://qaddemo:8080/qxi`
2. Select the Connections tab.
3. Expand the View Connection Pool menu.
4. Select the QADERP.FinAPI option from the menu.
5. Expand the Connections menu and select the All option.

You should see the connection pool with a single idle connection as shown below. If there are no idle sessions, your configuration is incorrect.



1.2. Deploy Financials API to QXI

The API that we are going to use for the lab is the Debtor Type API or Customer Type. This is a simple API and covers the basics of how Financials APIs are used by QAD QXtend. The Debtor Type API is one of the APIs that is installed with QXI, so you can simply add this API to the receiver. However, in this lab, let us try to manually add this API as a custom API. This applies to all APIs that are not installed with QXI.

Deploy the new Schema to QAD QXtend Inbound as a Custom schema

1. Open the QXI Web application in Internet Explorer: `http://qaddemo:8080/qxi`
2. Select the Configuration tab.
3. Select the Schemas node in the menu tree.
4. Select QADEE from the Schemas menu.
5. Click the Add button.
6. Choose the Continue Configuration update without suspending QXtend Inbound option and click the Submit button.
7. Set the following values:

   XML Syntax = Qdoc 1.1

   Route = Fin API Adapter

   Request Path = `<LabHomeDirectory>\Schemas\bdebtortype-ERP3_1.xsd`

   Procedure = `BDebtorType`



8. Deploy this API as a custom API.
9. Click the Next button.
10. Add the API to the QADERP receiver by selecting the check box and clicking the Done button.

1.2.1 Test the New API

The Debtor Type API has now been successfully deployed to QAD QXtend Inbound. However, until the new version has been used to process a couple of transactions, you cannot be sure that everything has been uploaded correctly.

The best way to test a QAD QXtend API is by using the soapUI product to build a WSDL Project and create some requests and process those requests using soapUI.

Generate the WSDL file first.

1. Open the QXI Web application in Internet Explorer:

   `http://qaddemo:8080/qxi`
2. Select the Configuration tab.
3. Select the Receivers node on the menu tree.

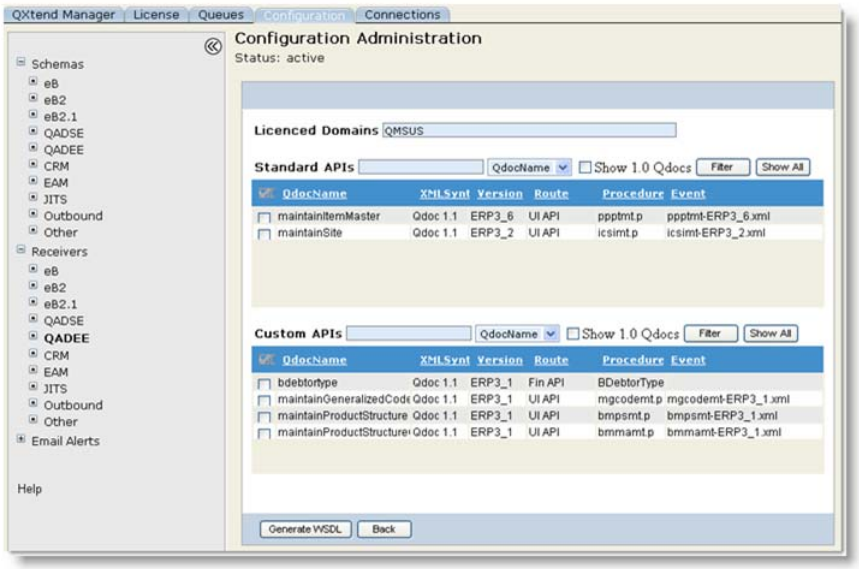
4 Select QADEE from the list of receivers.

5 Select the check box next to the QADERP receiver.

6 Click the View button.

7 Select bdebtortype from the list of Custom APIS.

8 Click the Generate WSDL button.



9 Click the No button for “Create backward compatible WSDL?”

10 Save `bdebtortype-ERP3_1.zip` to <*LabHomeDirectory*>

11 Unzip the file to <*LabHomeDirectory*>/`bdebtortype`.

Create the soapUI project.

1 Open soapUI on the Windows image using the shortcut on the Desktop.

2 Create a new workspace.

   a File – New Workspace.

   b Set the workspace name to `Lab 08 – Service Interface`.

   c Save the workspace file in <*LabHomeDirectory*>.

3 Create a new WSDL Project.

   a Right-click the workspace name.

   b Select the New soapUI Project option.

   c Set the project name to `Service Interface 08`.

   d Set the WSDL file to <*LabHomeDirectory*>/`bdebtortype/bdebtortype-ERP3_1.wsdl`

e Click OK to create the project.

1.2.1.1 Process a Create Transaction

Create a new Debtor Type in QAD EE

1 Drill down to the Request 1 message created under the Service Interface 08 project.

2 Right click Request 1 and select the Clone Request option. Enter a name for the new request.

3 Edit the new request so that it can be processed by QXI.

4 Change SOAP header:

a Set the receiver to `QADERP`.

b Set `suppressResponseDetail` to `true`.

5 Create the necessary session context entries. Create three `ttContext` iterations:

c Qualifier = `QAD`, Name = `version`, Value = `ERP3_1`

d Qualifier = `QAD`, Name = `entity`, Value = `10USACO`

e Qualifier = `QAD`, Name = `action`, Value = `save`

Note If you copy and paste the empty `ttContext` node and blank lines are created, right-click the request and select the Format XML option, soapUI will remove any blank lines and correct the indentation of the XML.

6 Edit the application data section of the message:

a The `BDebtorType` node by default will contain all of the fields available when processing a Debtor Type transaction. However, only a few fields are required for a transaction to process successfully. The best way to identify the mandatory fields is to run through the UI and see which fields you have to enter data for.

b Edit the `tContextInfo`:

- Remove all nodes except `tcActivityCcode`.
- You are creating a new Debtor Type so set the `tcActivityCode` = `Create`.

c Edit the `tDebtorType` information:

- Remove all nodes except `DebtorTypeCode`, `DebtorTypeDescription` and `DebtorTypeIsActive`. When processing a create using a Financials API, any values not provided in the request message will be defaulted.
- Set the following node values:

  `DebtorTypeCode` = `QXT1`

  `DebtorTypeDescription` = `QXtend Training Type`

  `DebtorTypeIsActive` = `true`
- Remove tTransString, tCustomTable0, tCustomTable1, and tCustomTable2.

7 Process the QDoc message that you have created in the new soapUI project and verify that the Debtor Type (Customer Type) is created successfully.

1.2.1.2 Process an Update Transaction

When QAD QXtend processes update transactions for a Financials API, if `tlPartialUpdate` in `tContextInfo` is set to false, then the request must contain all of the fields on the object that contain data—not just the fields required for the update to succeed.If a data element for the object is not included in the XML request to QAD QXtend, the Financials API will remove any data that was present in that field; this is also true of child iterations. For example, if a business relation has five addresses but you send in an XML with only one address defined, the other four addresses will be deleted. You must supply the full object and all data.

If `tlPartialUpdate` is set to true, then the request can only contain fields that required for the update (include the fields in primary key and fields need to be updated). The Financials API will keep original values for the fields that are not contained in the request.

To demonstrate this, you will modify the Debtor Type you created in Step 2.1.1:

1. Right-click the request created in Step 2.1.1 and select the Clone Request option. Enter a name for the new request.
2. Edit the application data section of the message:
   a. The `BDebtorType` node by default will contain all the fields available when processing a Debtor Type transaction. However, only a few fields are required for a transaction to process successfully. The best way to identify the mandatory fields is to run through the UI and see which fields you have to enter data for.
   b. Edit the `tContextInfo`.
      Remove all nodes except `tcActivityCcode` and `tlPartialUpdate`.
      You are modifying an existing Debtor Type so set `tcActivityCode` = `Modify`
      Set `tlPartialUpdate` to true.
   c. Edit the `tDebtorType` information (change it from active to inactive).
      Remove all nodes except `DebtorTypeCode` and `DebtorTypeIsActive`. When processing a create using a Financials API, original values for the fields that not provided in the request message will be kept.
      Set the following node values:

DebtorTypeCode	QXT1
DebtorTypeIsActive	false

3. Process the QDoc message that you have created in soapUI and verify that update in QAD EE.

1.2.1.3 Process a Delete Transaction

Create a delete request message:

1. Right-click the request created in Step 2.1.2 and select the Clone Request option. Enter a name for the new request.
2. Edit the application data section of the message: edit the tContextInfo and set the `tcActivityCode` to `Delete`.

3 Process the QDoc message that you have created in the new soapUI and verify that delete in QAD EE.

2. SI API

2.1 Create SI API Connection Pool

The FinAPI adapter connection pool establishes a connection to the Financials AppServer. Now we are going to create a SIAPI adapter which establishes a connection to the Native API AppServer. The new connection pool can be assigned to the same receiver as it is a different type.

2.1.1 Create New Connection Pool

Create an SI API connection pool for the QADERP receiver:

1 Open the QXI Web application in Internet Explorer: `http://qaddemo:8080/qxi`

2 Select the Connections tab.

3 Expand the Add Connection Pool menu.

4 Select the Add SIAPI Pool option.

5 Set the following values:

Pool Name = `QADERP` (must be identical to receiver name created in the previous step)

App Server Name = `qadsi_ASlive`

Host = `qaddemo`

Port (NameServer): `5162`

User = `qmi`

Password = “” (blank)

Domain = `10USA` (this is only the default domain, the domain can be passed with the request and will override this value on when processing the request).

Minimum Connections = `1`

Maximum Connections = `2`

6 Click the Save button to create the new connection pool.

2.1.2 Validate New Connection Pool

Now you need to check your configuration is correct and that you have idle sessions available.

1 Open the QXI Web application in Internet Explorer: `http://qaddemo:8080/qxi`

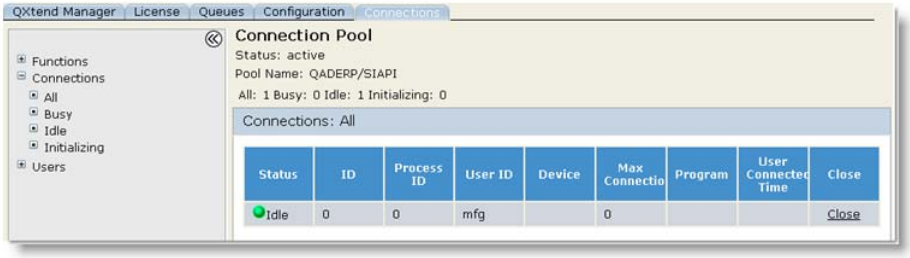
2 Select the Connections tab.

3 Expand the View Connection Pool menu.

4 Select the QADERP.SIAPI option from the menu.

5 Expand the Connections menu and select the All option.

You should see the connection pool with a single idle connection as shown below. If there are no idle sessions, your configuration is incorrect.



2.2. Adding APIs to a Receiver

1 Open the QXI Web application in Internet Explorer:

`http://qaddemo:8080/qxi`

2 Select the Configuration tab.

3 Select the Receivers node on the menu tree.

4 Select QADEE from the list of receivers.

5 Select the check box next to the QADERP receiver.

6 Click the Modify button.

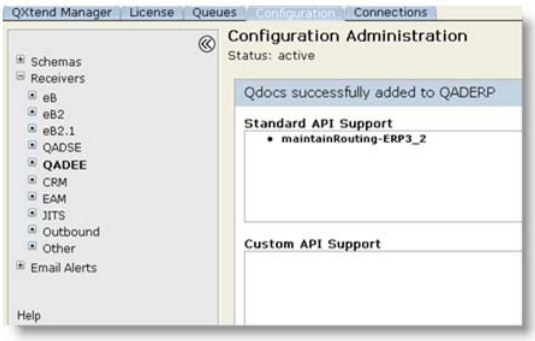
7 Choose the Continue Configuration update without suspending QXtend Inbound option and click the Submit button.

8 Click the Next button.

9 Click the Add QDoc button.

10 Locate the maintainRouting API and select the check box for the ERP3_2 version that uses the QDoc 1.1 XML syntax.

11 Click the Finish button. The receiver update report displays.



2.3 Test the New API

2.3.1 Generate the WSDL file.

1 In QXI|Configuration, select the Receivers node on the menu tree.

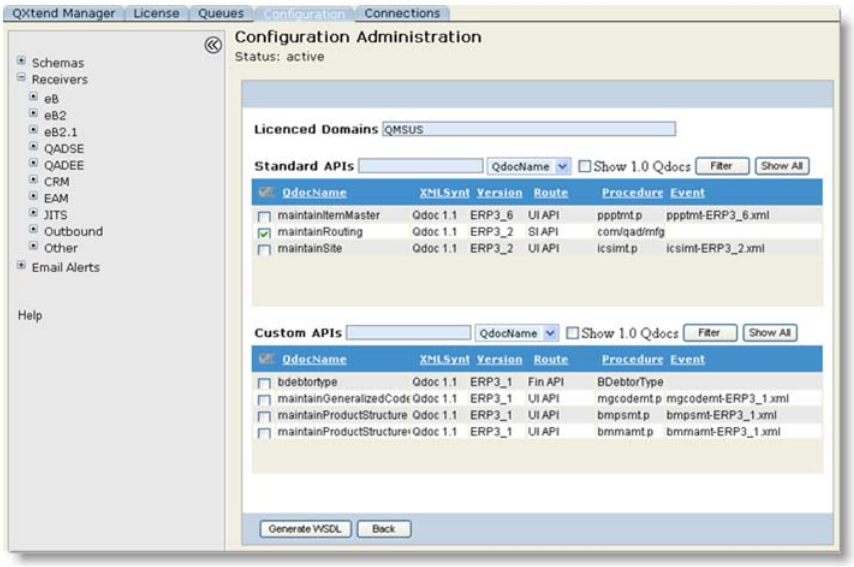
2 Select QADEE from the list of receivers.

3 Select the check box next to the QADERP receiver.

4 Click the View button.

5 Select maintainRouting from the list of Standard APIs.

6 Click the Generate WSDL button.



7 Click the No button for “Create backward compatible WSDL?”

8 Save maintainRouting-ERP3_2.zip to <*LabHomeDirectory*>.

9 Unzip the file to <*LabHomeDirectory*>/maintainRouting.

2.3.2 Create a soapUI project

1 Open soapUI on the Windows image using the shortcut on the Desktop.

2 Use the existing workspace `Lab 08 - Service Interface-workspace`.

3 Create a new WSDL Project.

- **a** Right-click the workspace name.
- **b** Select the New soapUI Project option.
- **c** Set the project name to maintainRouting.
- **d** Set the WSDL file to <*LabHomeDirectory*>/maintainRouting/maintainRouting-ERP3_2.wsdl

e Click OK to create the project.

2.3.3 Process a Create Transaction

Create a new routing maintenance in QAD EE

1 Drill down to the Request 1 message created under the maintainRouting project.

2 Right click Request 1 and select the Clone Request option. Enter a name for the new request.

3 Edit the new request so that it can be processed by QXI.

4 Change SOAP header:

a Set the receiver to `QADERP`.

b Set `suppressResponseDetail` to `true`.

5 Create the necessary session context entries. Create three `ttContext` iterations:

c Qualifier = `QAD`, Name = `version`, Value = `ERP3_2`

d Qualifier = `QAD`, Name = `domain`, Value = `10USA`

Note If you copy and paste the empty `ttContext` node and blank lines are created, right-click the request and select the Format XML option, soapUI will remove any blank lines and correct the indentation of the XML.

6 Edit the application data section of the message:

a The `deRouting` node by default will contain all of the fields available when processing a Routing Maintenance transaction. However, only a few fields are required for a transaction to process successfully. The best way to identify the mandatory fields is to run through the UI and see which fields you have to enter data for.

b Edit the XML and create a new Routing record, use the following values:

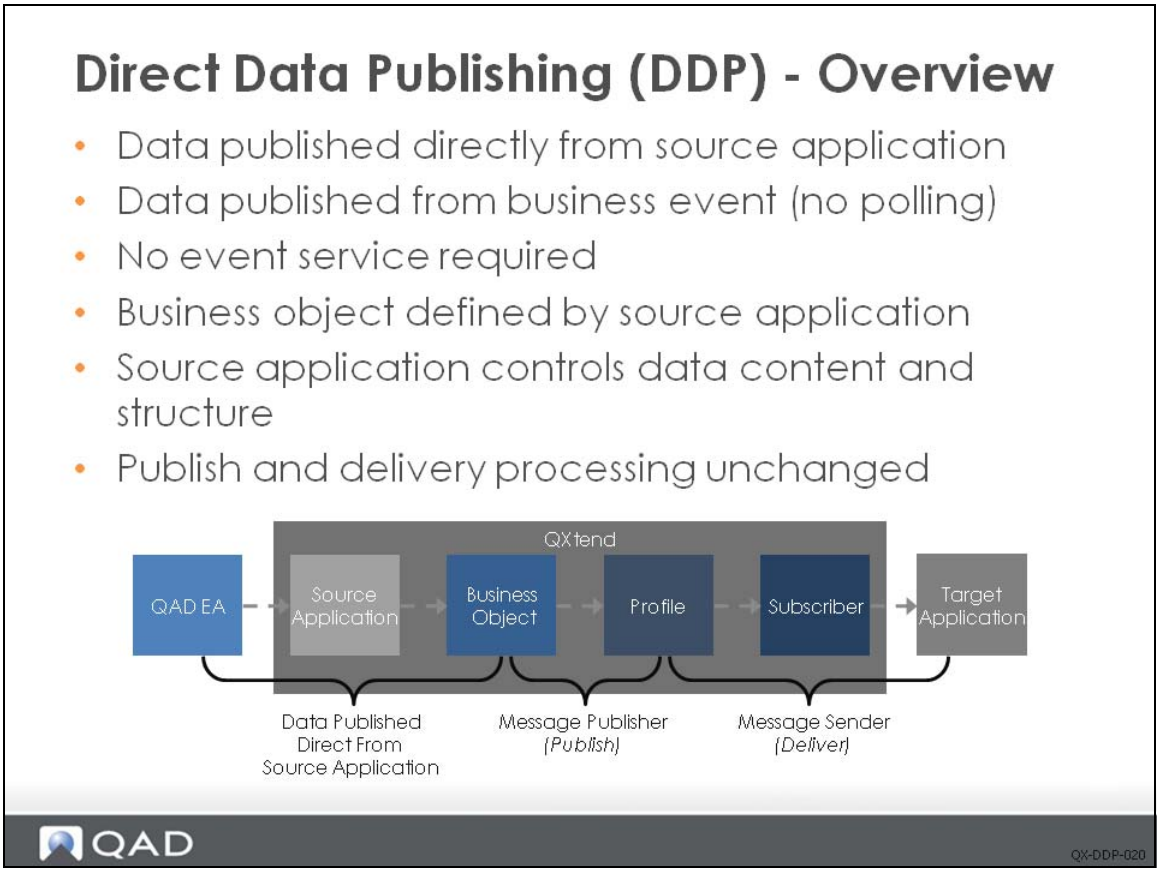
- `Routing.operation=A`
- `routing.roRouting = q1000`
- `routing.roOp=10`
- `routing.roStart=2010-12-01`
- `routing.roWkctr=1000`

7 Process the QDoc message that you have created in the new soapUI project and verify that the Routing record is created successfully.

Chapter 9

Direct Data Publish

Overview



QXO supports two modes of operation for a source application: Data Extraction and Direct Data Publish (DDP).

In Data Extraction mode QXO is responsible for extracting business object data after a business event has occurred; the extraction process is orchestrated by the event service. Data Extraction is the most widely used operating mode and works with any Open Edge database. However, this approach has limitations; for example, the fact that data is published after the business event occurs means that the business object data could change during the time between when the event occurred and when the data was extracted.

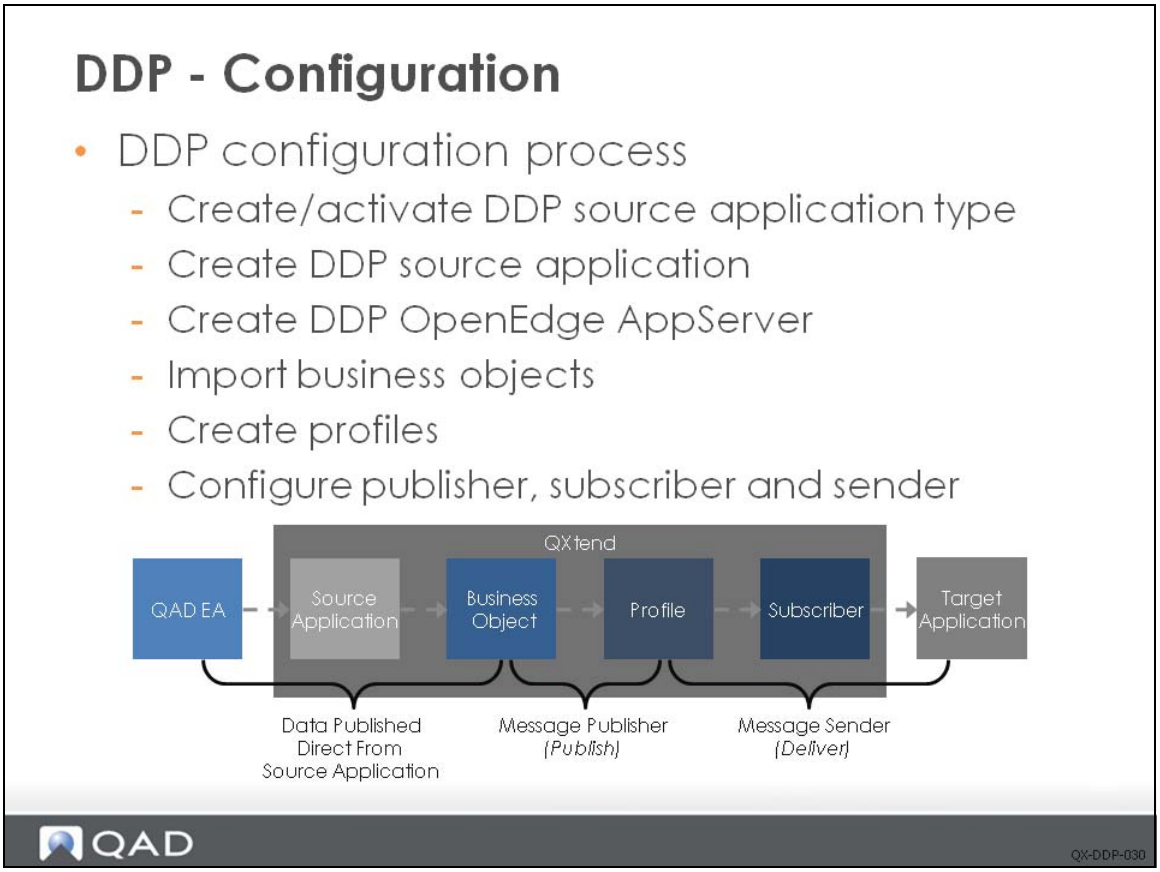
In Direct Data Publish (DDP) mode, the source application is responsible for getting business object data related to a business event to QXO for processing—QXO does not extract the data. The source application must identify a business event, package the relevant business object data, then pass it to QXO using the DDP API provided by QXO. The DDP API can be called by connecting to a OpenEdge AppServer or by calling the QXI Web service to access the API. The content of the business object published is controlled completely by the source application and can be the result of complex business logic. DDP provides a unique mechanism for collating data to be published to external applications; Data Extraction mode cannot provide this same mechanism since the data is limited to data in the database.

Since QXO is not responsible for extracting data from the source application, the event service is not required for source applications that use DDP. Currently a source application cannot use both Data Extraction and DDP modes. When configuring a DDP source application, you do not have to configure a set of databases, since QXO does not need to connect to the source application database to poll for events and extract data.

With Data Extraction source applications, the business object is user-defined. However, with DDP the business object is defined by the source application: the business object is imported into QXO (rather than being user-defined) and its structure cannot be changed. You create business objects for a DDP source application by importing an XML schema that defines the business object.

Once the data is published, a business object instance is created and the Publish and Deliver phases of the QXO process are executed, using the same rules regardless of the operating mode of the source application. If changes to the data content are required for the subscriber, these changes must be made in the profile definitions for the business object.

Configuring DDP in QAD QXtend



Configuring QXtend for DDP is similar to configuring it for Data Extraction; the major differences are in the set up of the source application and the creation of business objects.

To configure DDP within QXO, do the following:

1. Create/activate DDP source application type. A source application can operate in either a Data Extraction or Direct Data Publishing mode, which is reflected in the configuration of the source application type. In the source application type screen, ensure that the DDP check box is selected. If the DDP source application type has already been defined, make sure it is active by selecting the Active check box.

2. Create DDP source application. A DDP source application instance must be created before data can be sent to QXO using the DDP API. Create a new instance from the source application tree menu: assign a name to identify the data source. Once the source application instance has been created, business objects can be created.

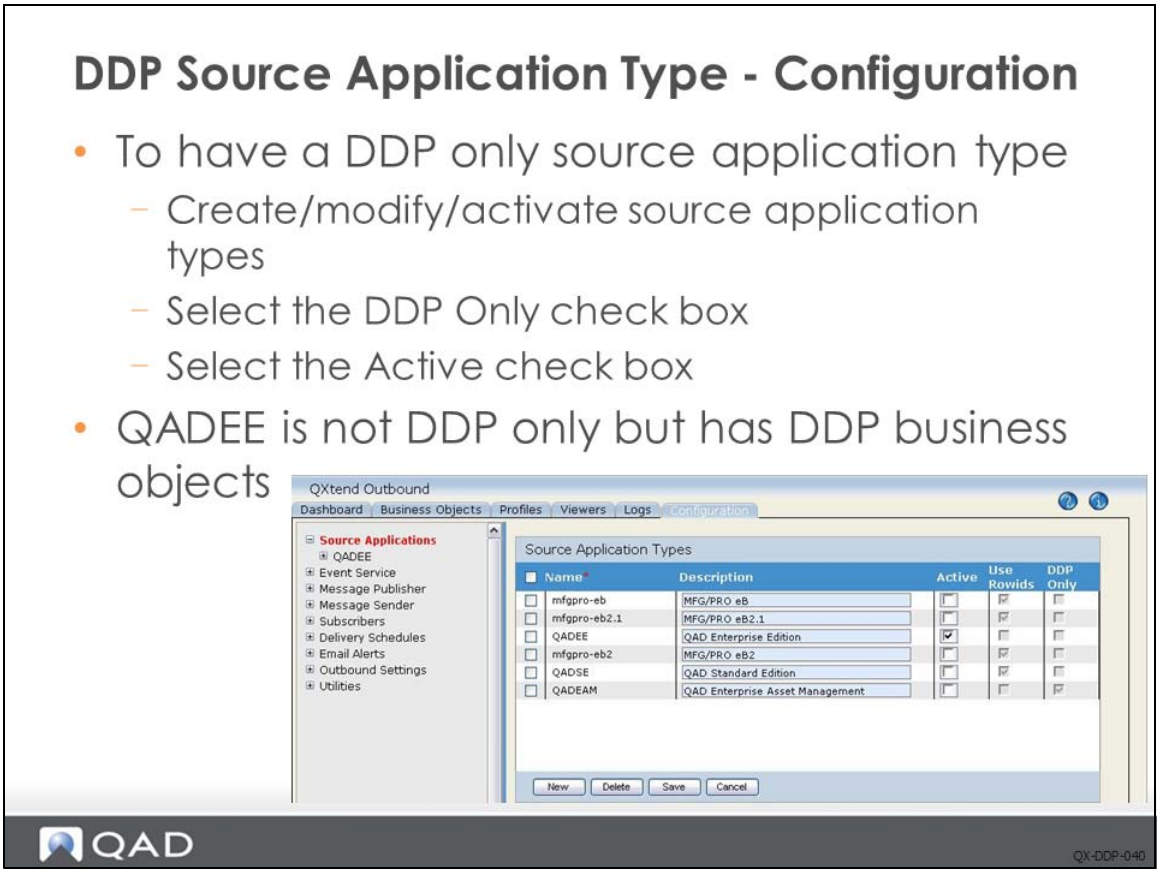
3. Start DDP AppServer. To enable the source application to post data to QXO using the DDP API, an AppServer instance must be running that connects to the QXO database. The QXO runtime library must be in the PROPATH and the instance must have the QXO service interface startup procedure configured. This AppServer is used by the source application to invoke the DDP API.

4 Import business objects. Business object definitions can be created on the Business Objects tab. First you select a source application type that is a DDP type. New business objects can be created; instead of the system prompting you to build the definition manually, you are asked for an XML schema file that defines the business object. The DDP source application must supply the business object definition XML schema files.

5 Create profiles. Defining profiles for DDP source application business objects is performed the same way for both operation modes of the source application; fields can be renamed, excluded, and additional calculated field can be added. All profile functionality applies to all business objects regardless of the operating mode of the source application.

6 Configure QXO services. Make sure that the remaining QXO services are correctly configured to ensure that the relevant data is published to interested subscribers. You will need to configure the message publisher, subscribers, and the message sender as described in the *QXO Configuration* section.

Configuring the DDP Source Application Type.

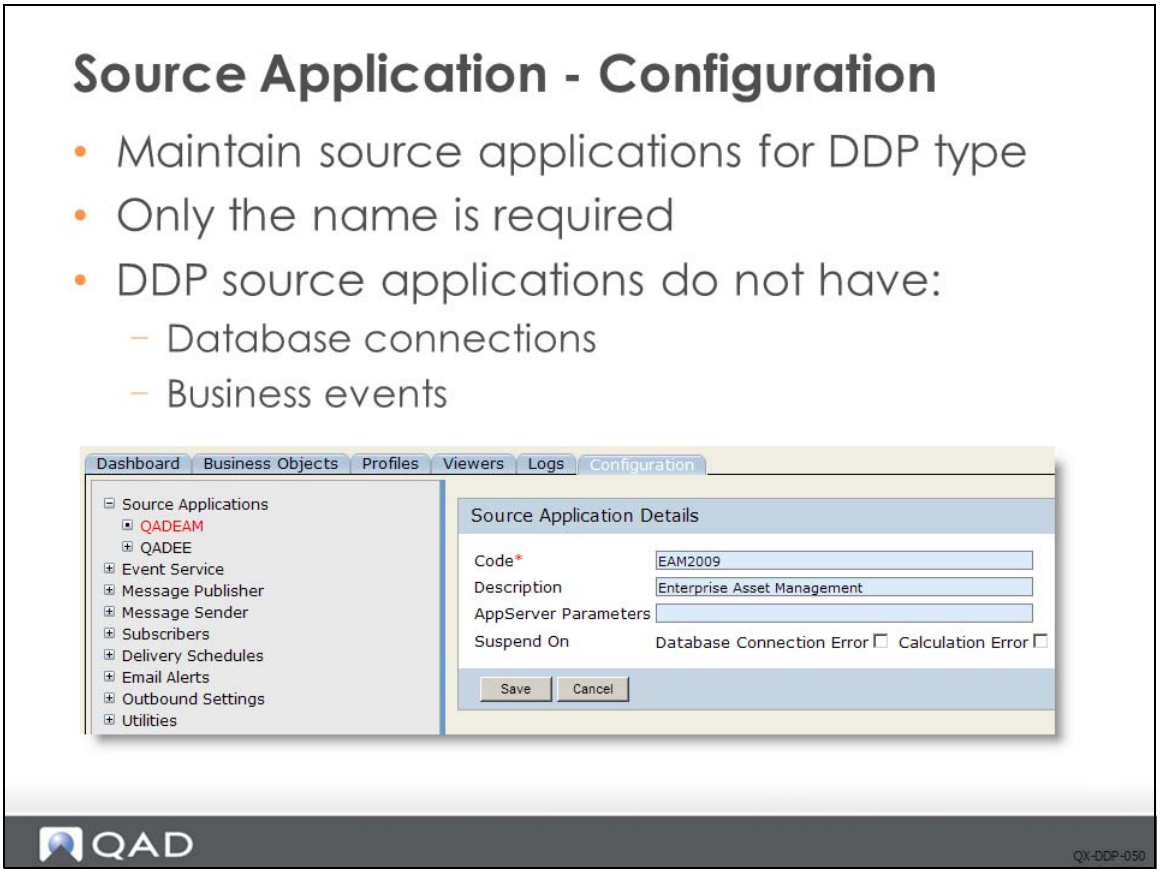


Selecting the Source Applications node in the tree menu on the Configuration tab in QXO displays the current source application types. Use this screen to maintain the source application types available to QXO.

Ensure that a DDP source application type has the corresponding DDP Only check box selected; this controls the behavior of QXO when managing business objects. Also, when using a source application type, make sure that the Active check box is selected; otherwise, it will not be available in other areas of QXO. Each active source application types is displayed as a menu option under the Source Applications node.

All business objects of DDP only source application type are DDP business objects, but DDP business object can also be created in source application types that are not DDP only. QADEE is the typical source application type which is not DDP only but has DDP business objects (Financials).

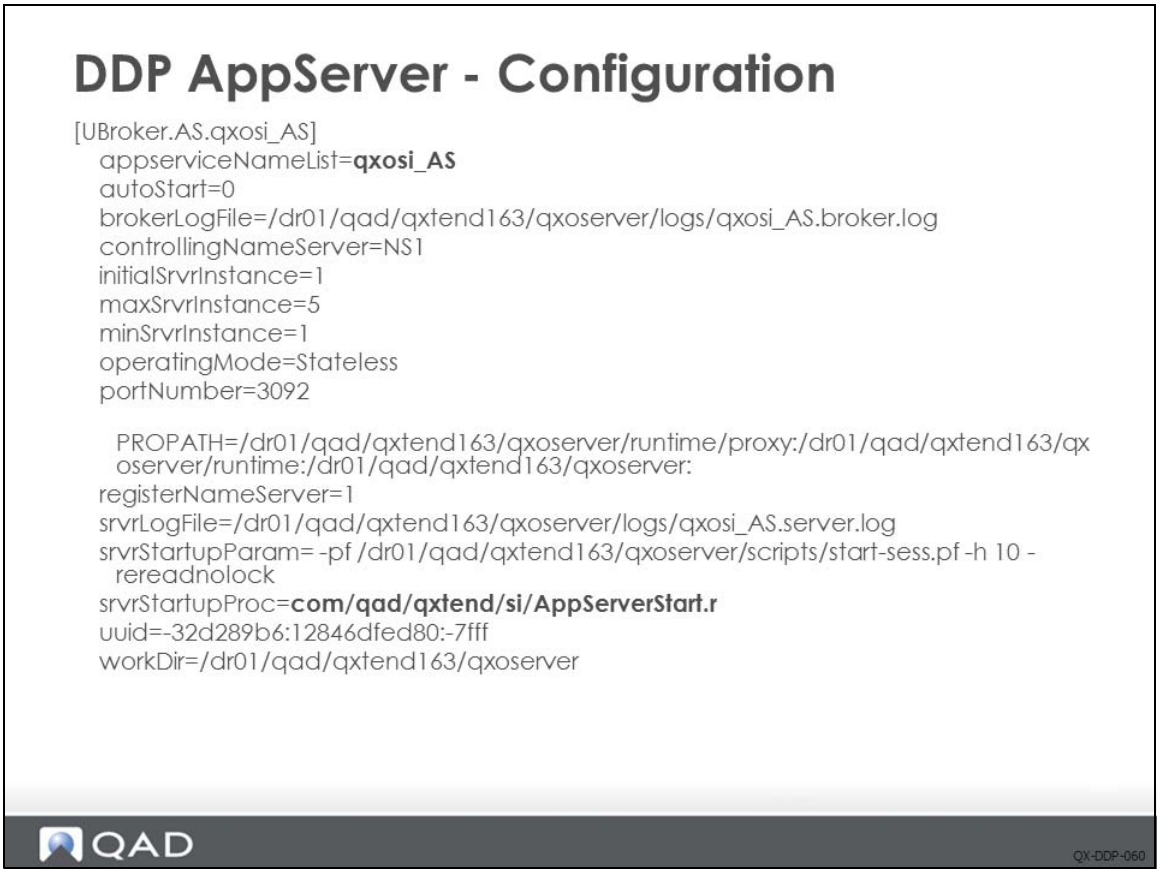
Configuring the DDP Source Application



Select the DDP source application type from the Source Application node in the tree menu to display the source application instances. Click New to create a new instance. The name for the source application is the only information required for a DDP source application. However, the name is important because when data is published to QXO, the sending application must supply the name entered here so that QXtend knows which instance the data is for.

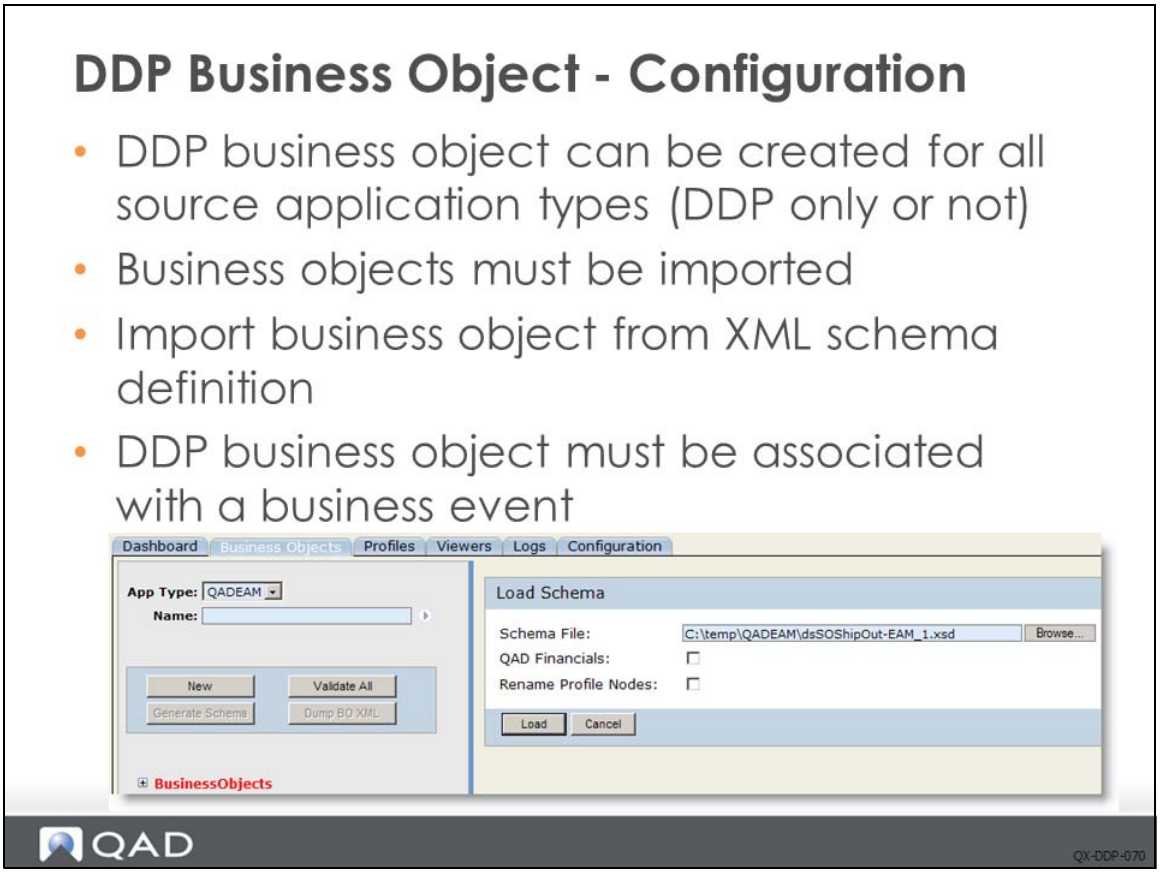
You do not have to define a set of database connections for DDP source applications because the database is used by the event service and DDP does not use the event service. This is also true of events: DDP source applications do not use business events in the current release.

Configuring the DDP AppServer



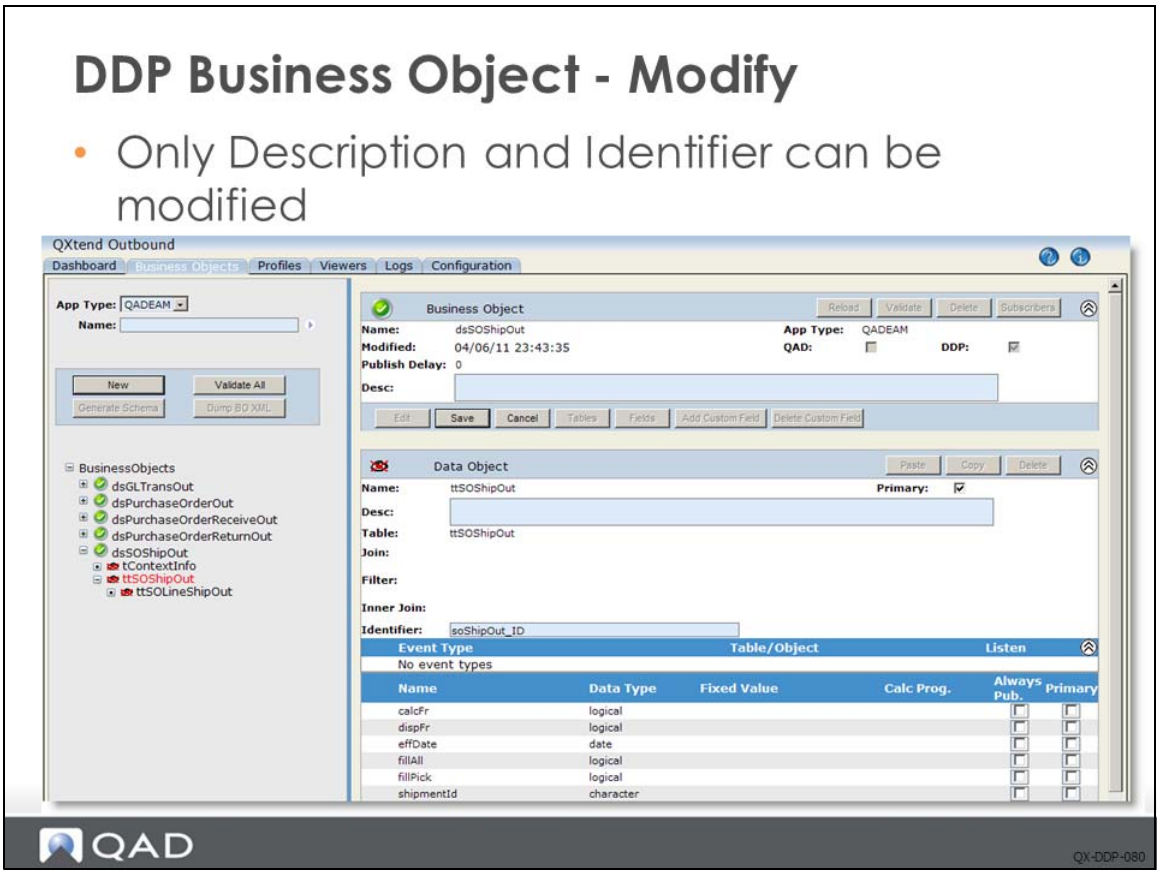
DDP source applications publish data to QXO via the DDP API, which is a QXO service. In order to call this service, an AppServer instance that is connected to the QXO database and QXO runtime code is required. The AppServer must also be configured to load the QXO SI startup procedure `com/qad/qxtend/si/AppServerStart.r` when it starts. Typically the AppServer is set up during installation. This AppServer needs to be running in order for DDP source applications to be able to publish data to QXO.

Configuring DDP Business Objects



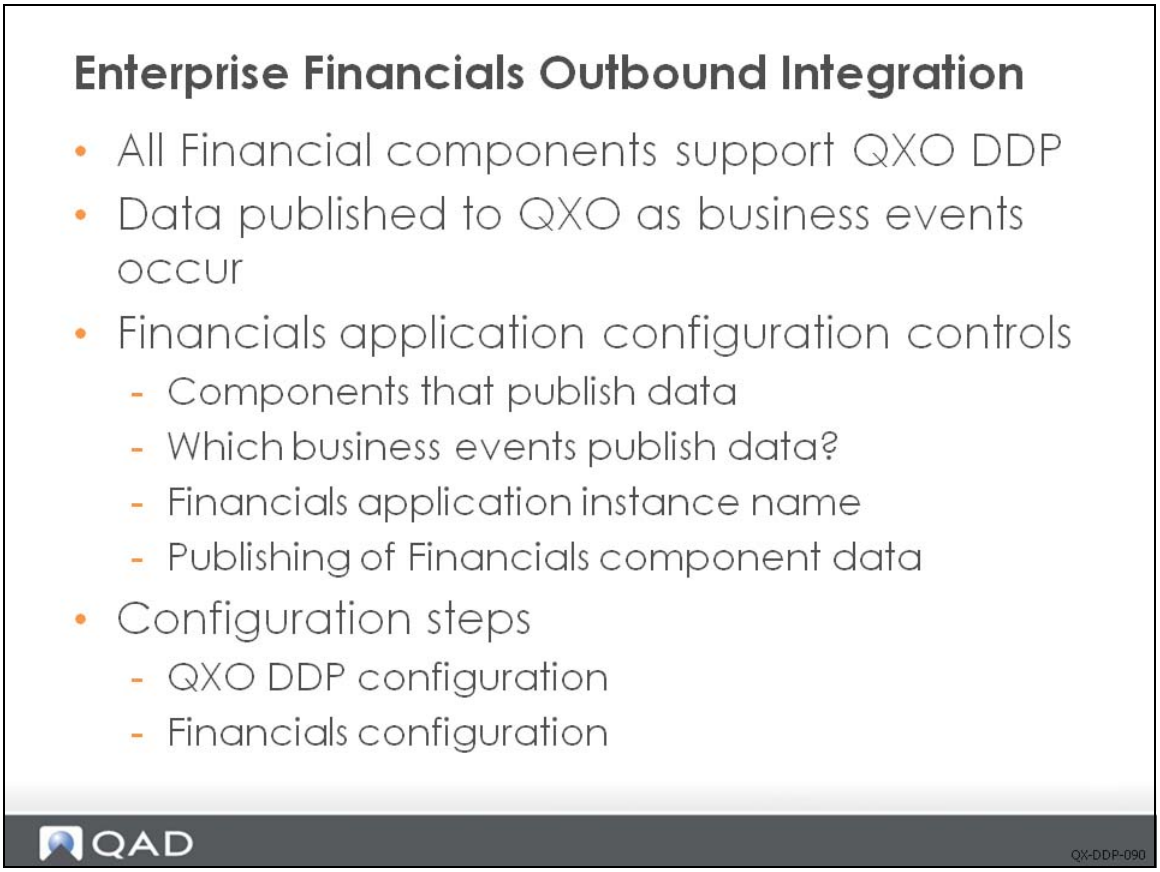
DDP Business objects cannot be manually created or modified—they must be imported from an XML schema definition file. The source application is responsible for supplying the business object definitions. These definitions must be available on the local machine creating the business objects.

Click New and navigate to the schema definition file for the business object; click Load to load the file. This creates a business object that can be reviewed like any other business object. Only the Description and Identifier fields can be modified.



The Identifier field is used to define which fields in a data object to use to define a unique identifier for that business object instance in the source application.

Enterprise Financials Outbound Integration



The introduction of the Enterprise Financials that are part of QAD Enterprise Edition brings with it a new implementation architecture that is built around object-oriented development principles. As part of this development approach, the Financials application consists of business components—for example, Debtor, Creditor, Debtor Invoice and so on.

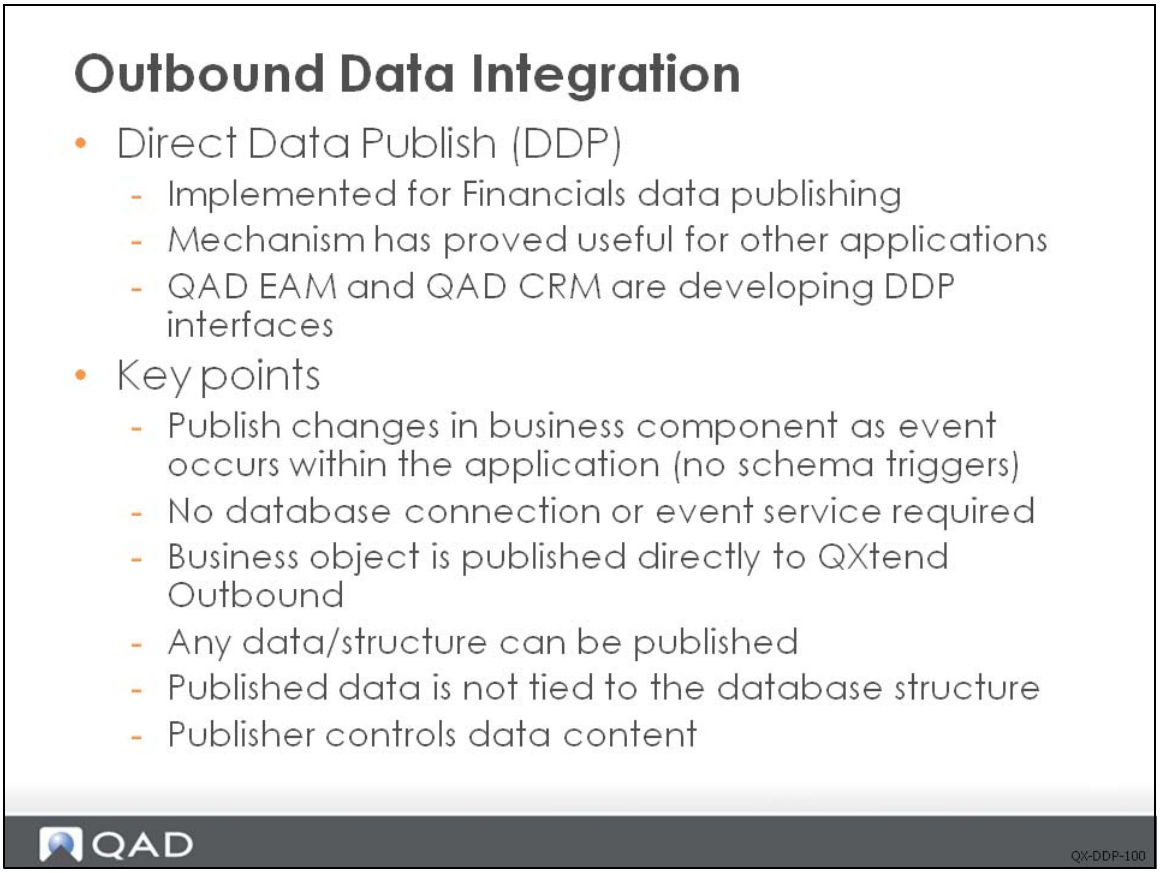
The business components each have their own lifecycle and events occur against these business objects. As the business event occurs, all of the data relevant to the current state of the business component is available to be published to applications that are interested in the business event. The ability to publish these business events to interested applications is built into the foundation of QAD EE, any financial component can publish business event data.

The event publishing mechanism is used to publish data to QXO using DDP. The advantage of this approach is that any data published reflects the state of the component as the event occurred—there is no time lag between the event occurring and the extraction.

You can configure Enterprise Financials to control the data obtained, the business events the data is obtained from, and the location to where data is published.

To publish Enterprise Financials data using QXtend requires you to configure QXtend and Enterprise Financials. The following sections describe the required configuration.

Direct Data Publish

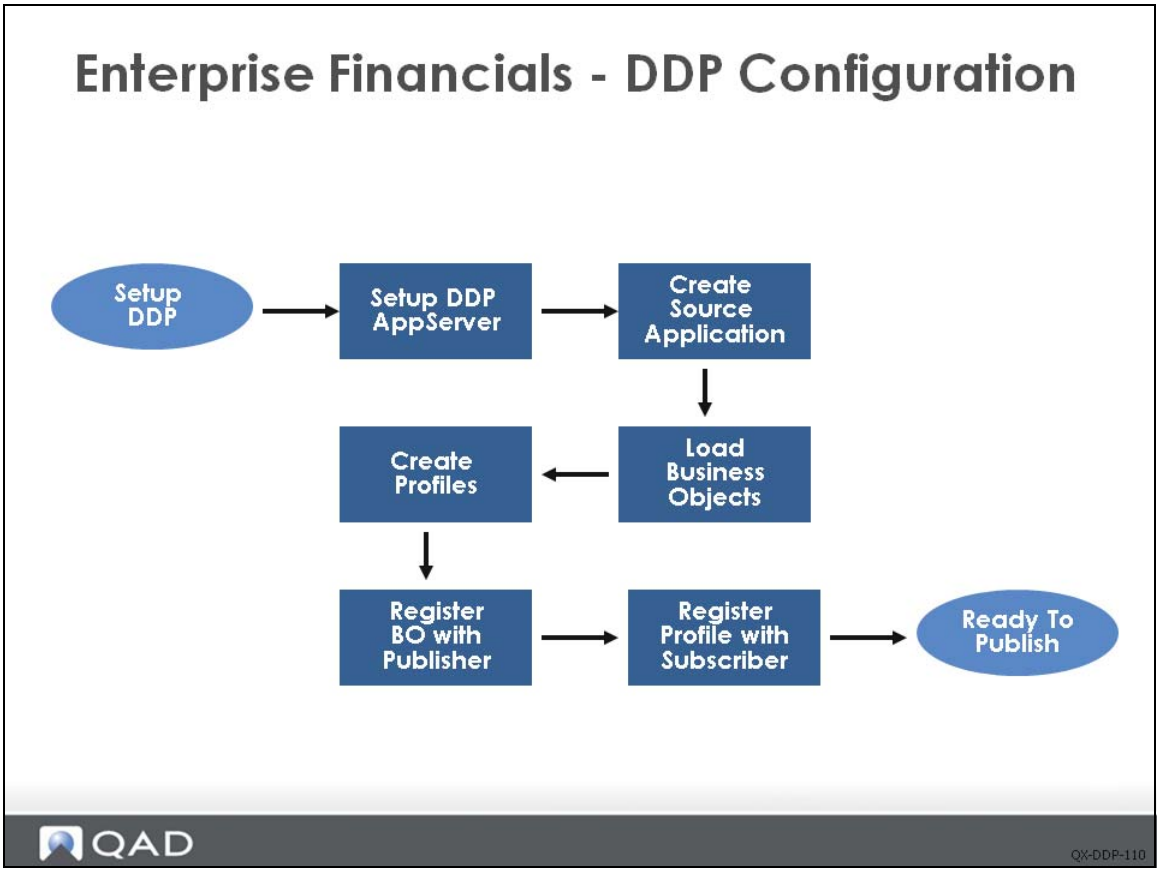


A key requirement of Financials is to be able to synchronize data from one instance of the Financials module to another, or to other external systems. To support this requirement, it is essential to provide an infrastructure that allows data to be published from and loaded into the Financials module. The QXtend framework already provides infrastructure to support data synchronization between QAD Application domains. To ensure that customers have a single integration and synchronization solution, QXtend will be enhanced to support data synchronization for the Financials module.

The Financials module has been engineered differently to QAD EA. It supports true business object functionality and recognizes state changes of the object throughout its lifecycle. Therefore there is no need to extract data from the Financials database, QXtend only needs to be able to accept the raw business object event data and publish it.

QXtend is the interoperability framework of choice for future QAD projects, and other applications might need to publish data. The design needs to be generic enough to allow other applications to plug in to QXtend in the same manner as the Financials module.

Enterprise Financials - DDP Configuration

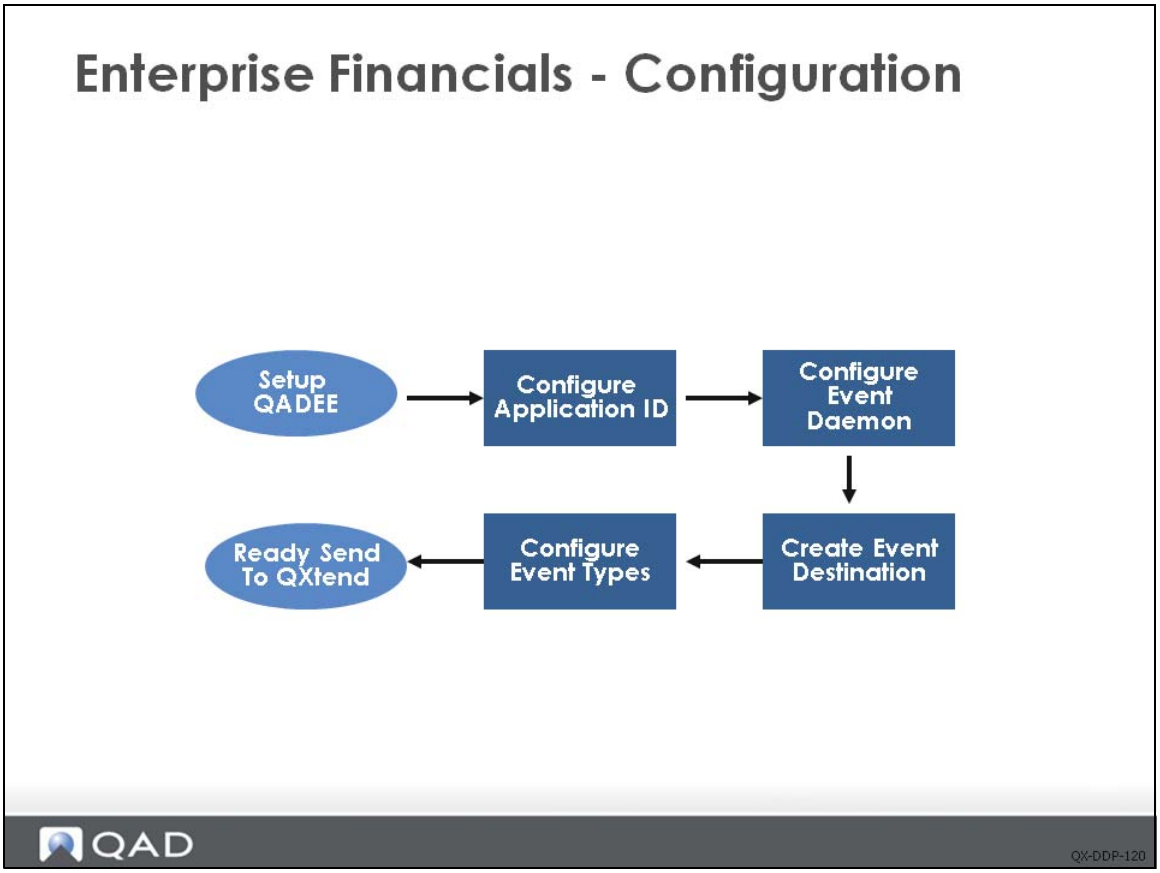


Enabling data publishing to QXtend from QAD Enterprise Financials requires setup in both Financials and QXO. The setup for QXO is standard configuration for DDP. To configure QXO for DDP, do the following:

1. Setup DDP AppServer. An OpenEdge AppServer must be configured that is connected to the QXO database, with the QXO Service Interface startup procedure enabled. Usually this is done during the installation of Qxtend.
2. Create source application. Ensure that the QADEE type is active, and then use it to create a new source application instance. As we usually do both data extraction and direct data publish for QADEE, so set the source application as a normal one (include database connections). Events must be imported for this source application and corresponding business events need to be activated.
3. Load business objects. The business objects you want to publish from QAD Enterprise Financials must be imported for the QADEE source application type. Use XML Import to load schemas for QADEE.
4. Create profiles. The profiles that define the format of the outbound data from QXO to the subscribers need to be created for each of the business objects being published.
5. Register business object with message publisher. For the messages that get published from the Financials, the business object must be registered with a message publisher instance.

6 Register profile with subscriber. Associate profiles with the relevant subscribers. The subscriber must already be registered with a message sender instance.

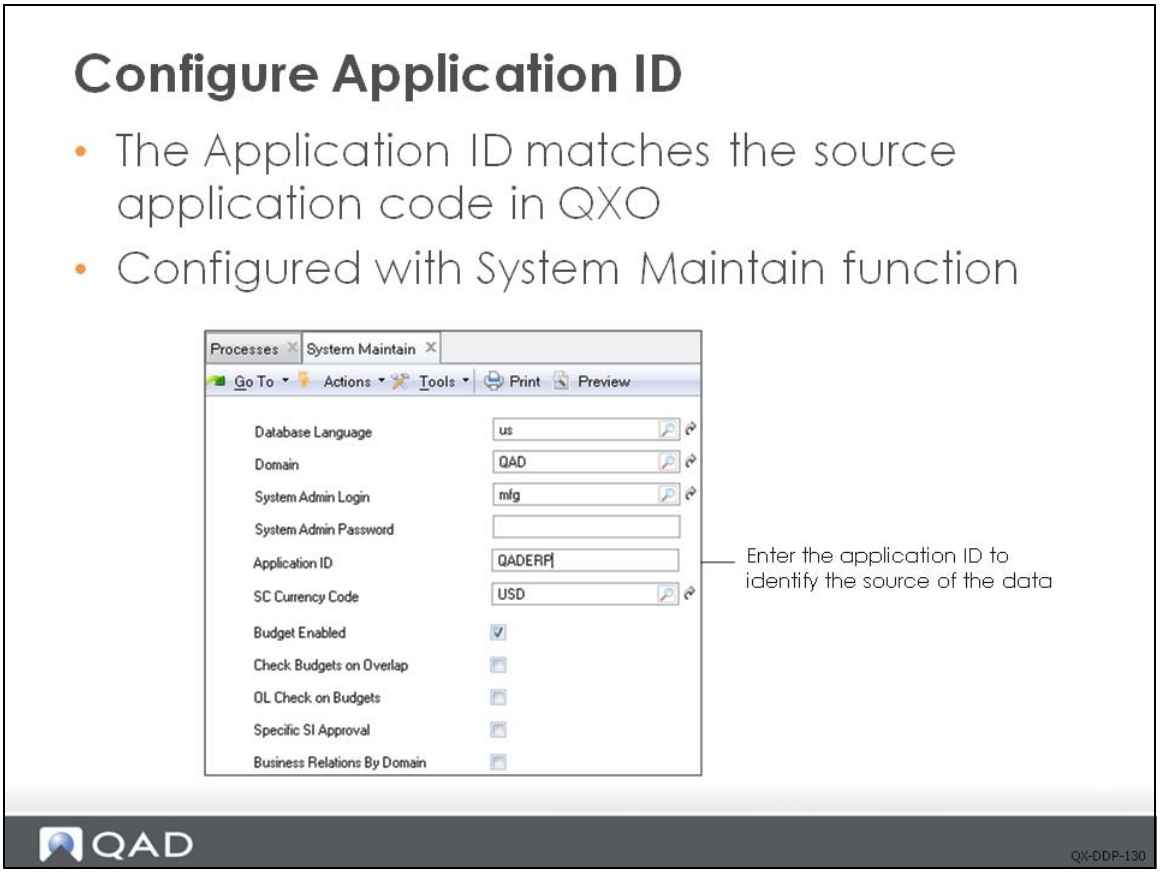
Configuring Enterprise Financials for DDP



Enabling Enterprise Financials to publish data to QXtend requires the following steps to be performed within Enterprise Financials:

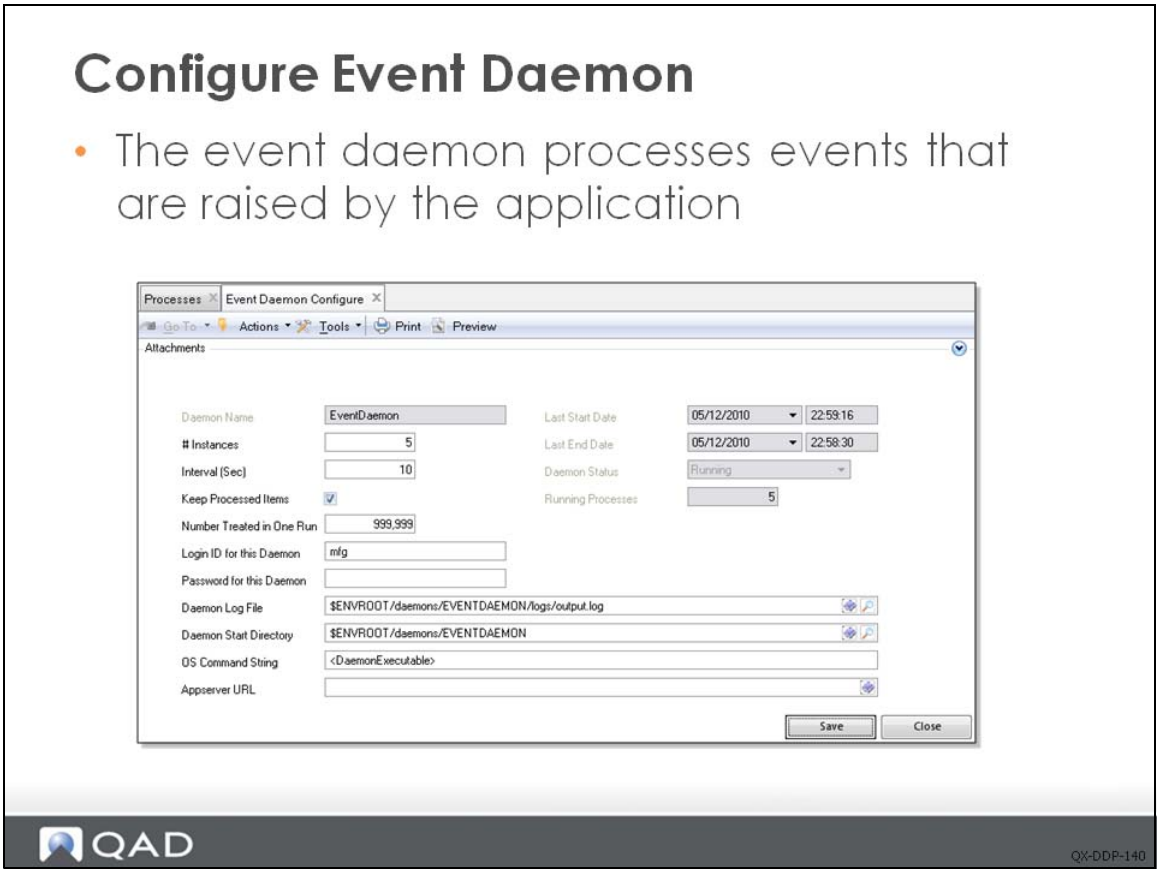
1. Configure application ID. Identifies the Enterprise Financials instance. The application ID must be the same name given to the source application created in QXO.
2. Configure event daemon. The event daemon is a background process that runs against the Financials instance and is responsible for publishing data to QXO. For data to be published, an event daemon needs to be configured.
3. Create event destination. Defines the target to which financial data will be sent; for QXtend this is the DDP AppServer configured during QXO configuration.
4. Configure event types. These link events raised against the Financials business components to the event destination created in Step . This controls which business events send their data to QXO.

Configuring the Application ID



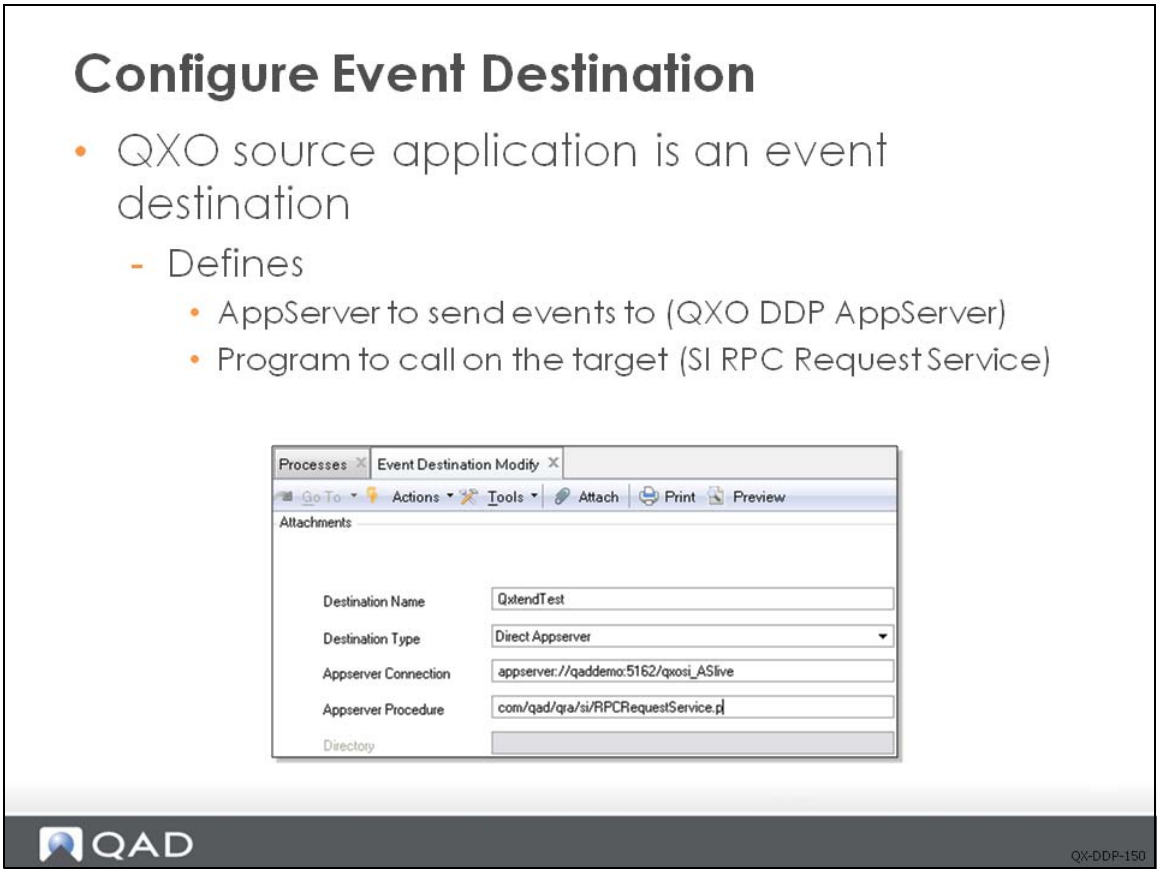
The System Maintain function allows you to maintain a set of global system settings. One of those settings is the Application ID; this field is included in the message that is published to QXO. QXO uses this value to identify the source of the data, so it must be set to the same value as the name of the corresponding source application in QXO.

Configuring the Event Daemon



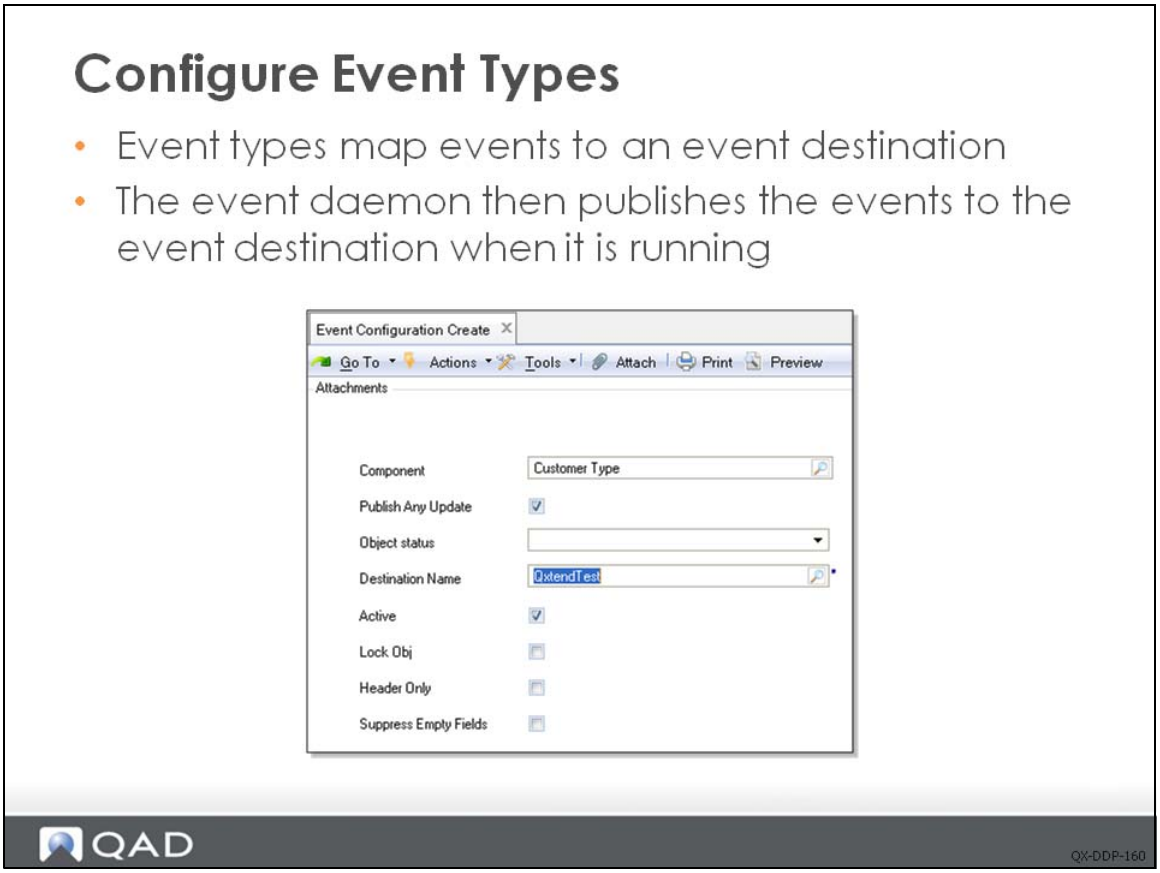
For details see QAD EE documentation.

Configuring the Event Destination



The event daemon created in the previous step processes the business events and delivers them to event destinations. QXO needs to be configured as an event destination. A new event destination needs to be created and the Appserver connection configured to connect to the QXO DDP AppServer in the QXO DDP configuration. Finally, the Appserver procedure needs to be set to `com/qad/qra/si/RPCRequestService.p`. This configuration will be used as the connection to QXO when publishing data.

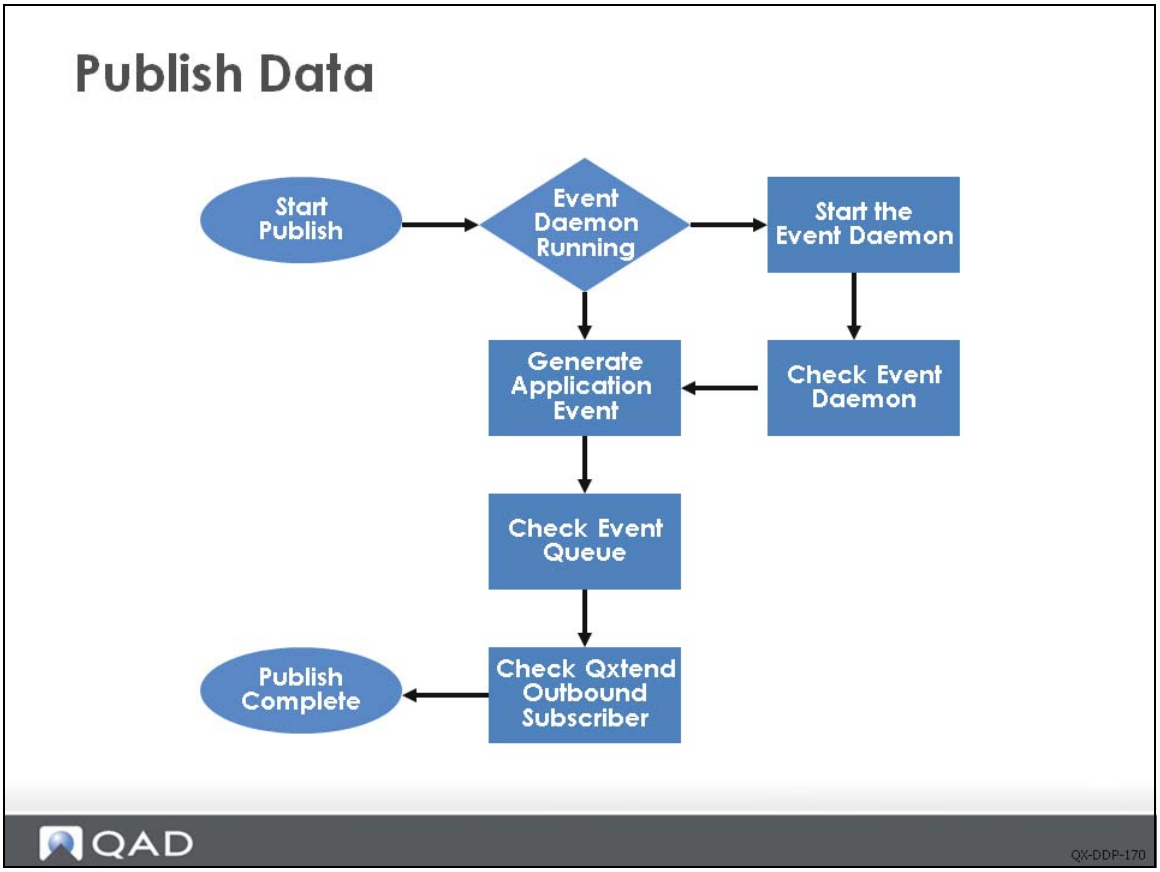
Configuring the Event Types



The Event Configuration screen in Financials allows you to create a subscription to a business event within Enterprise Financials. The business component that will raise the event is selected, and the QXtend destination created in the previous step is assigned.

When business events are raised against the component, the event data is published to the QXO destination, which ultimately will publish the data to QXO using QXO DDP.

Publishing the Data

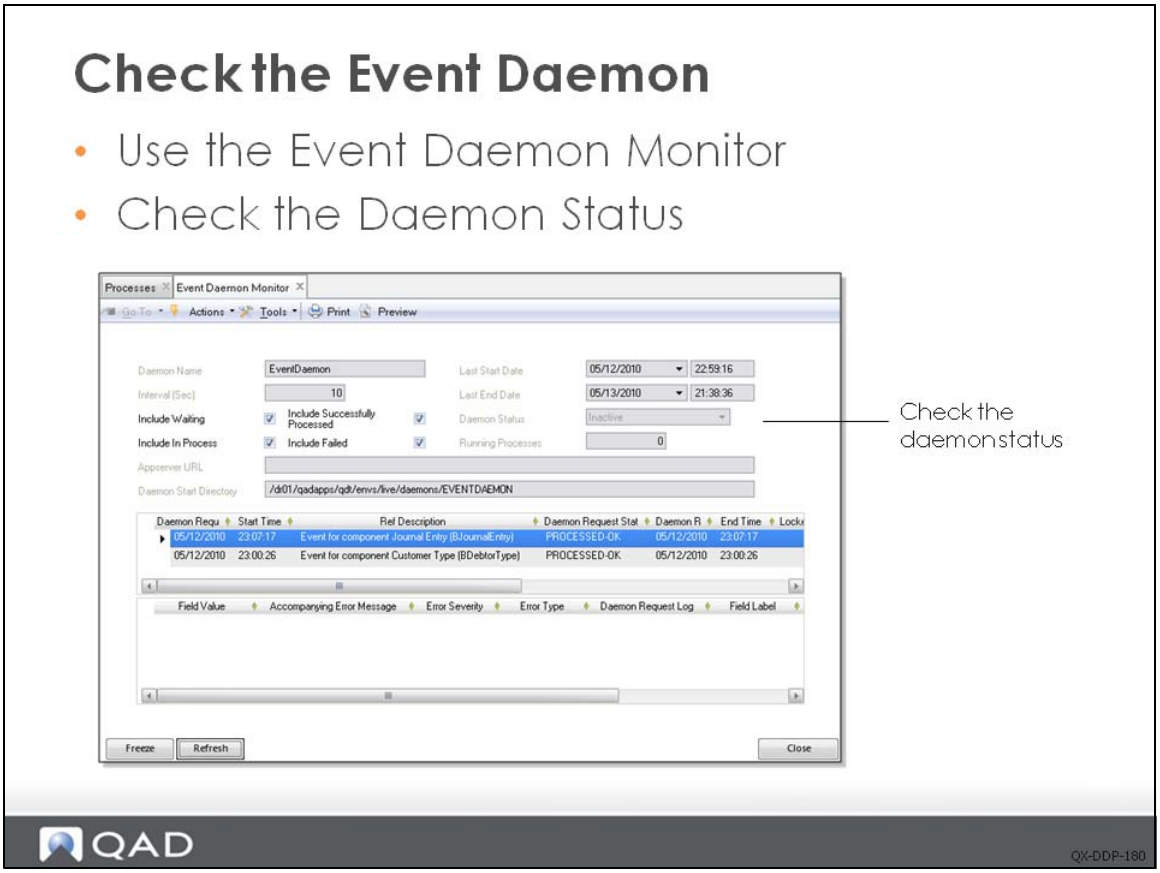


To publish data from QAD Enterprise Financials, a transaction must be processed against one of the business components that has been configured to send data to QXO. However, before creating the transaction, make sure that the event daemon is running. To check the event daemon, use the Event Daemon Monitor function. If the daemon is not running, start it by using Event Daemon Start.

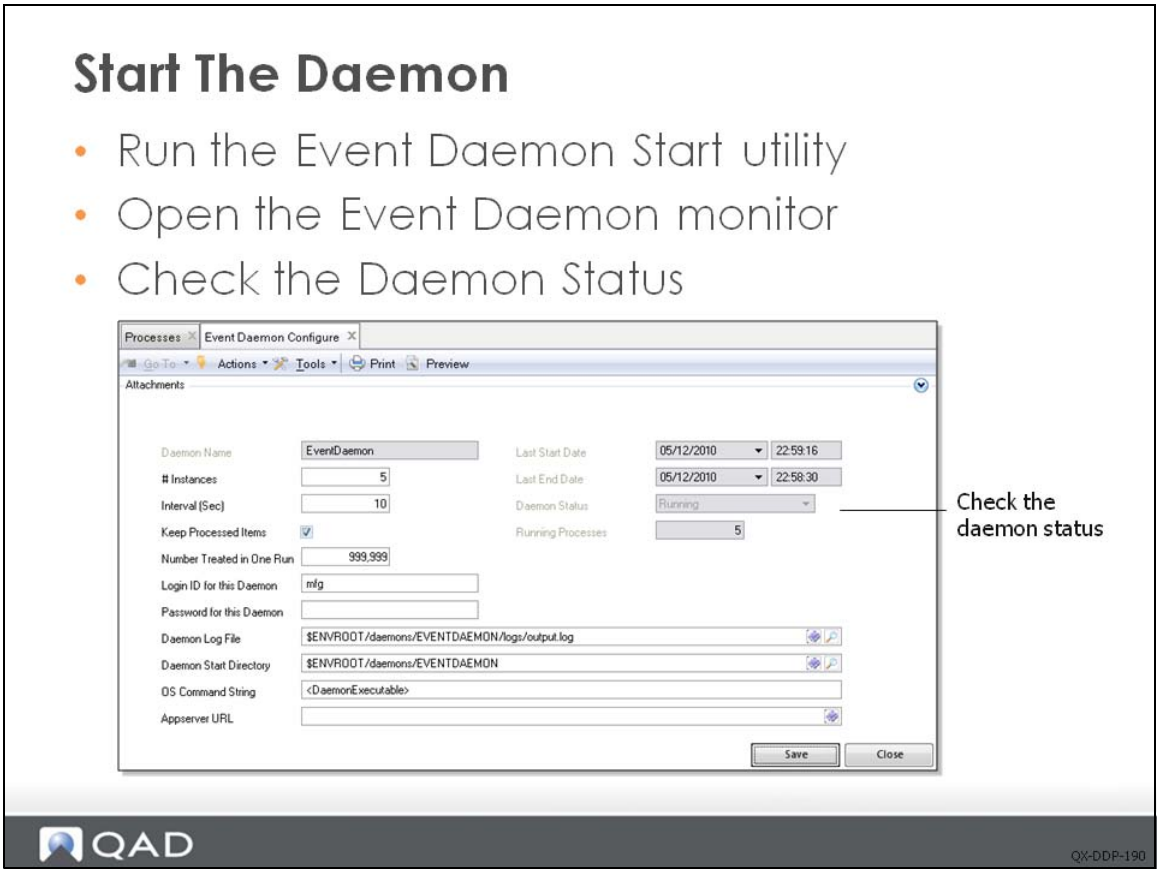
Create a new entry in the business component configured to publish data to QXO; this will generate an event message.

Using the Event Monitor function, check the event queue. You will see the message that was raised by the application and published to QXO. Then go to QXO and look in the Subscriber Messages viewer. You will see a new message that contains the data that has been published by Financials.

Checking the Event Daemon



Starting the Daemon

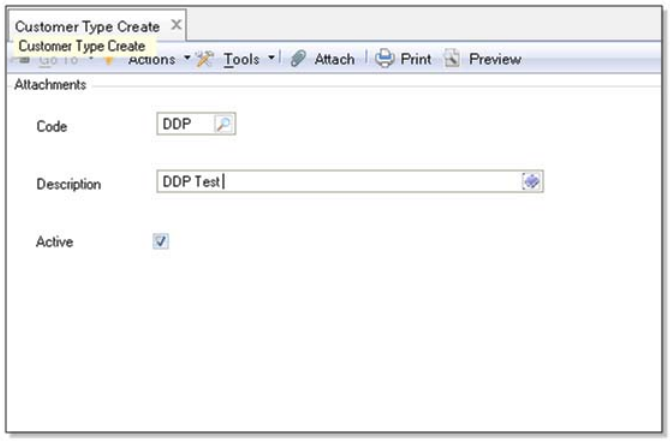


You can use Event Daemon Start to start the daemon, and use Event Daemon Stop or Event Daemon Unconditional Stop to stop the daemon. You can also use Event Daemon Clear Queue to remove the messages in queue.

Generating an Application Event

Generate Application Event

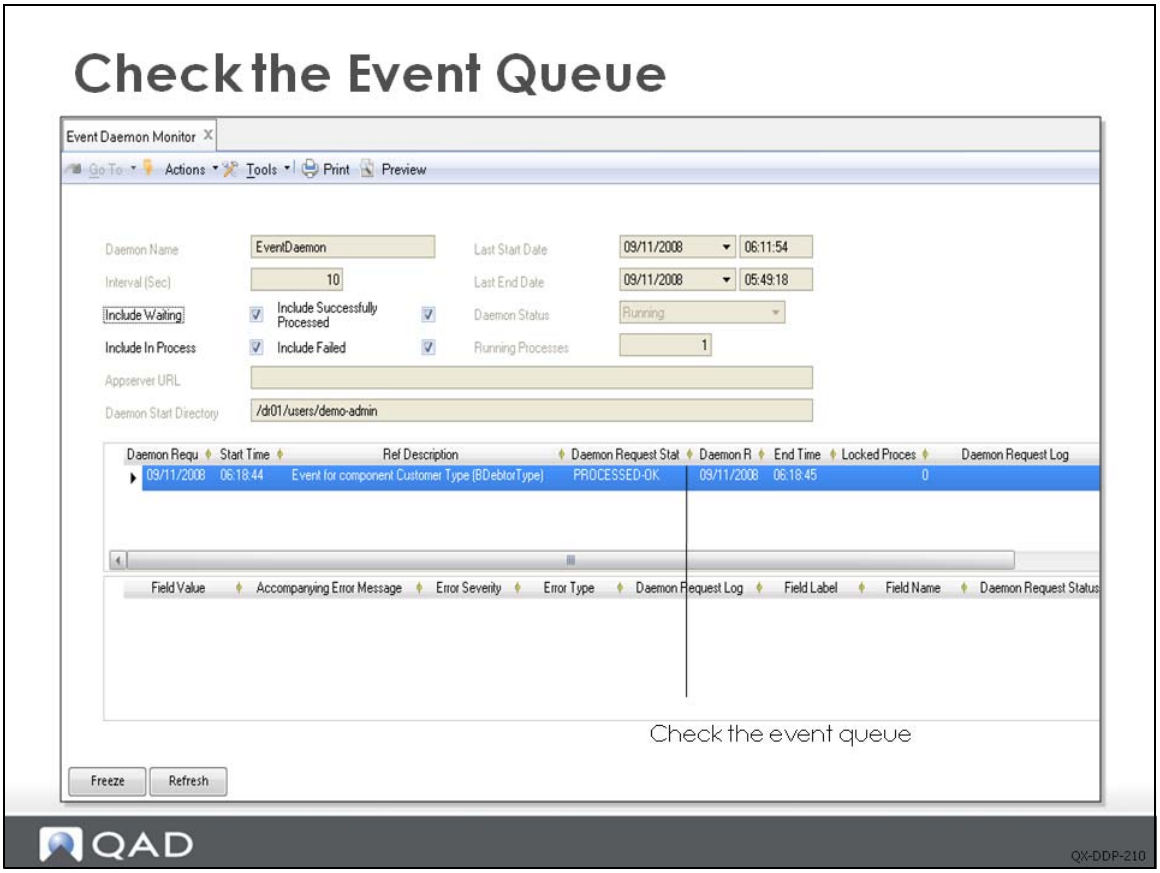
- Create a monitored event



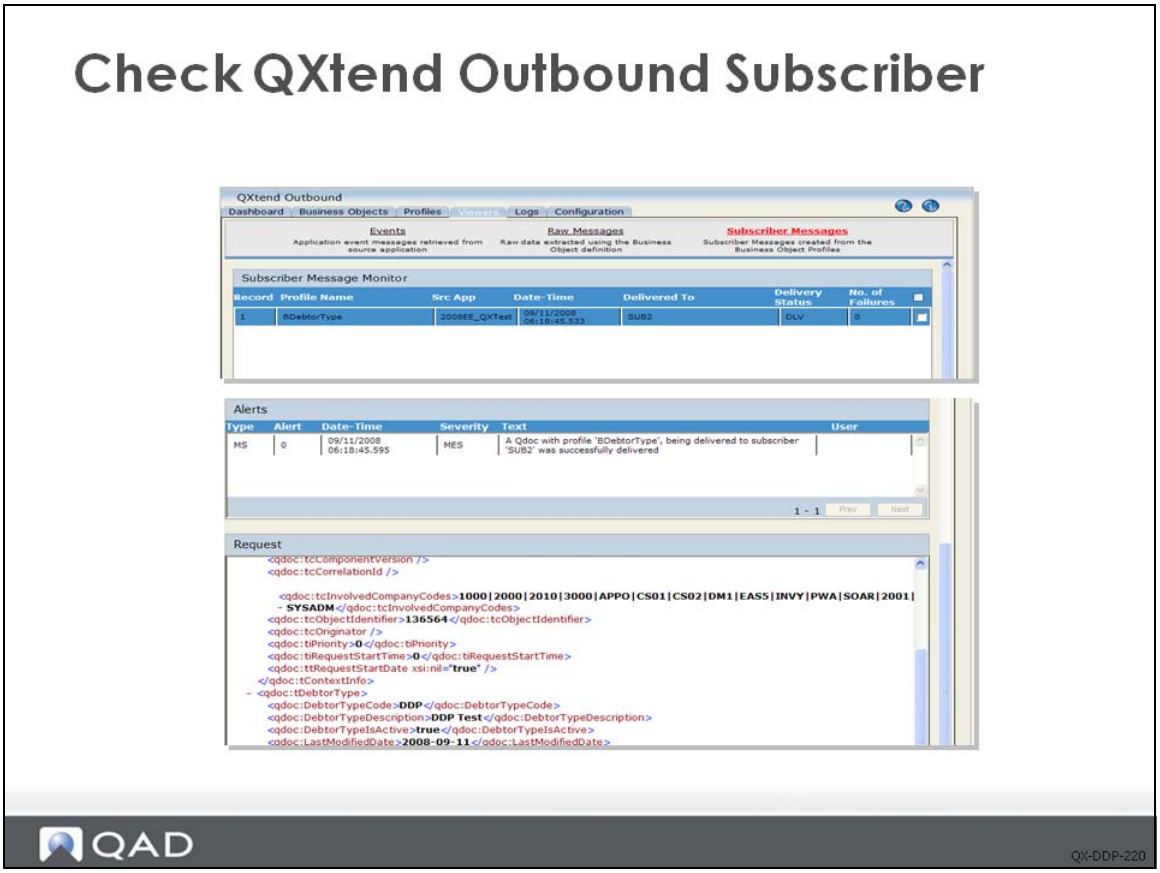
QAD

QX-DDP-200

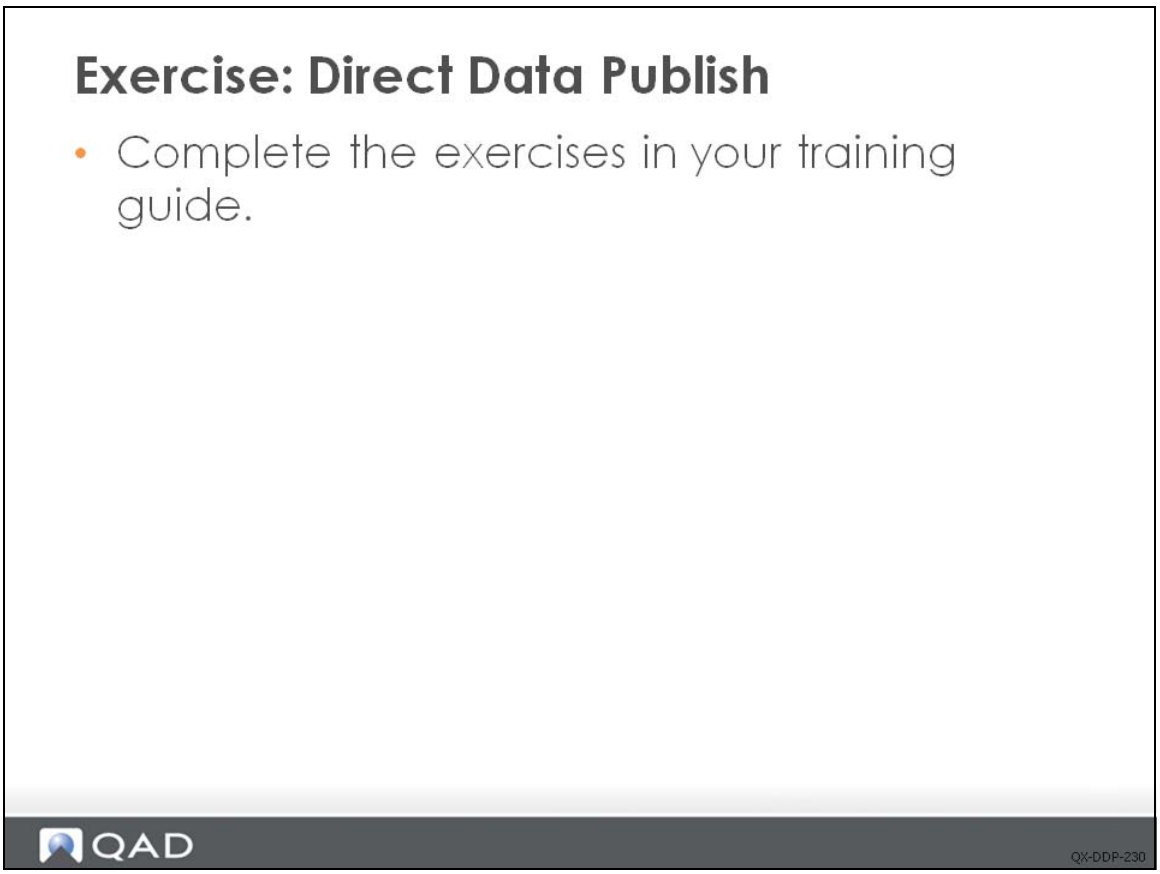
Checking the Event Queue



Checking the QXO Subscriber



Exercise: Direct Data Publish



The following list shows a number of key terms and concepts for DDP. In each statement below, fill in the correct term from the list. Some terms are used more than once.

event service	Direct Data Publish (DDP)
event service	timelag
Data Extraction(DE)	source application
event destination	event configuration
database connections	business event
Application ID	source application

1 QXO supports two modes of operation for a source application: _____ and _____.

2 In data extraction mode QXO extracts business object data after a _____ has occurred; extraction is orchestrated by an _____. In DDP mode, the _____ is responsible for getting business object data related to a business event to QXO for processing. Consequently you do not have to configure an _____ when using DPP.

3 You do not have to define a set of _____ for DDP source applications because DDP does not use an event service.

4 One of the advantages of using DDP is that there is no _____ between when an event occurred and when the business object is published and delivered.

5 When configuring QAD EE for use with DPP, you use the System Maintain program to enter the _____, which identifies the source of the data. This must be set to the same value as the name of the corresponding _____ in QXO.

6 _____ defines the target to which financial data will be sent; while _____ link events raised against the Financials business components to the _____ .

Lab: Direct Data Publish

QAD QXtend 1.7.2 has been installed with both QXI and QXO components. In the previous labs you performed the basic configuration that is required to enable requests to be processed by QXI and QXO. In the following lab exercises you will see how to publish data direct from a source application using Direct Data Publish. The QAD EE Financials module supports DDP and you will use it in this exercise to publish data to QXO.

`<LabHomeDirectory> = C:\QXtendTraining\Labs\09-DDP\`

The configuration for the publishing of data from the QAD Financials module is a two-step process:

1. Configure QXO so that it is ready to receive data.
2. Set up QAD EE so that it knows to publish to QXO.

This lab describes both configuration steps.

1. Configure QXO for Financials DDP

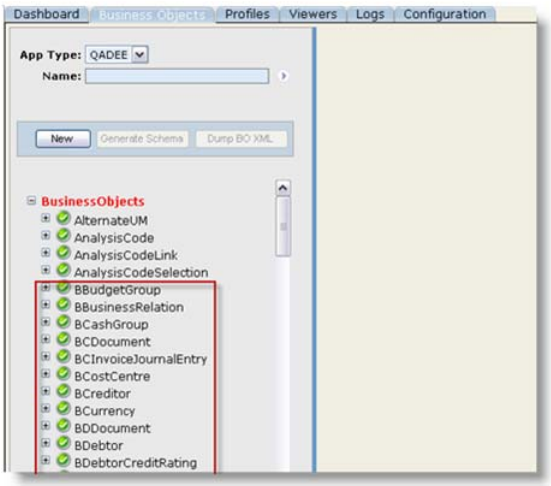
The following step must be followed to configure QXO so that it is ready to receive data published to it directly from the QAD Financials module.

1.1 Create Source Application

It has been done in *QXO Configuration*. The Source Application QADERP has been created.

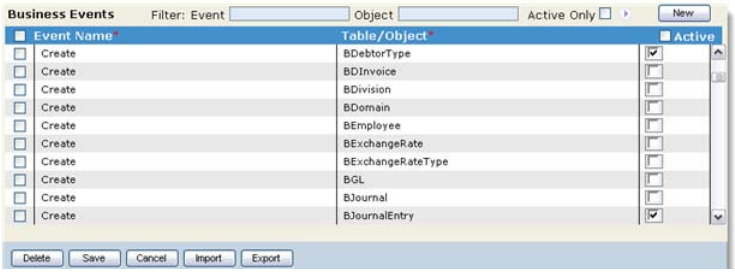
1.2 Load Business Object Definitions

Business Object Definitions have been loaded in the lab for *QXO Configuration*. All Financials Business Objects start with ‘B’.



1.3 Activate Event Types

1. Open the QXO Web application in Internet Explorer: `http://qaddemo:8080/qxo`
2. Select the Configuration tab.
3. Select the `Source Applications\QADEE\QADERP\Event Types` menu option.
4. Under Business Events, click the Active checkbox of Create event of BDebtorType and BJournalEntry.



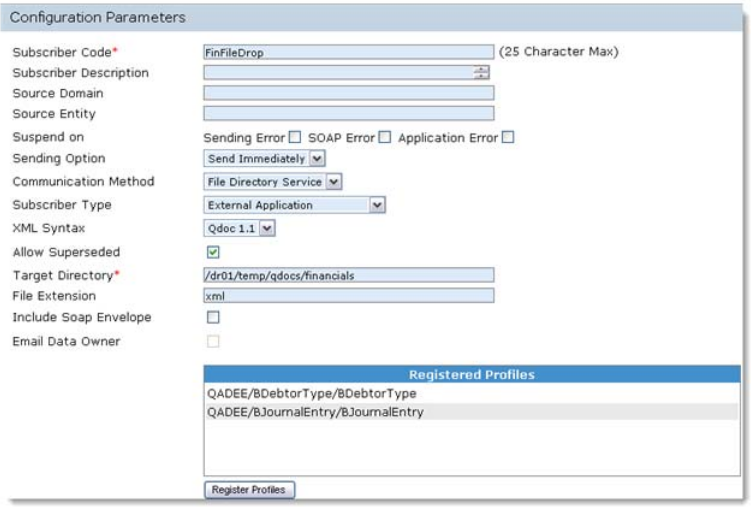
5. Click the Save button.

1.4 Create QXO Subscriber

QXO can publish the DDP data sent to it from a source application to any of the supported delivery methods. In this lab we are going to create a new subscriber that subscribes to the data published from QAD EE, and then delivers that data using a file drop to a directory on the server running QXO.

Create the Subscriber

1. Open the QXO Web application in Internet Explorer: `http://qaddemo:8080/qxo`
2. Select the Configuration tab.
3. Select the Subscribers menu option.
4. Click the New button.
5. Set the following values:
   - Subscriber Code = `FinFileDrop`
   - Communication Method = File Directory Service
   - XML Syntax = QDoc 1.1
   - Select the Allow Superseded check box
   - Target Directory = `/dr01/temp/qdocs/financials`(make sure the directory exists and writable)
   - Register the BDebtorType and BJournalEntry profiles with the subscriber



6 Click the Save button.

1.5 Configure QXO Services

Now that you have created a subscriber for the data, we want to publish from QAD EE. We now have to configure the services in QXO to make sure that any data sent to QXO is published and delivered. The QAD EE Financials module uses DDP to publish data direct to QXO, so we do not need to change the event service as this process is bypassed when using DDP. You must change the message publisher and message sender services to ensure that data from the Financials module flows through QXO.

1.4.1 Update the Message Publisher

The configured message publisher within QXO needs to be updated to publish data for the two business objects.

1 Open the QXO Web application in Internet Explorer: `http://qaddemo:8080/qxo`

2 Select the Configuration tab.

3 Select the Message Publisher menu option.

4 Select the MP1 menu option.

5 Add the BDebtorType and BJournalEntry business objects from the QADEE source application type to the list of registered business objects for the message publisher.

6 Click the Save button.

1.4.2 Update the Message Sender

Two message senders are configured in QXO. For messages to be delivered to the FinFileDrop subscriber created in Step 1.3, we need to add it to one of these message sender services.

1 Open the QXO Web application in Internet Explorer: `http://qaddemo:8080/qxo`

2 Select the Configuration tab.

3 Select the Message Sender menu option.

4 Select the MS2 menu option.

5 Register the FinFileDrop subscriber with the Message Sender.

6 Click the Save button.

1.4.3 Stop and Start QXO Services

QXO caches the configuration information that drives each of the services and when changes are made to the service configuration all of the QXO services need to be stopped and restarted for the changes to come into effect.

1 Open the QXO Web application in Internet Explorer: `http://qaddemo:8080/qxo`

2 Select the Dashboard tab.

3 Select the Services menu option.

4 Click the Stop All button.

5 Once all of the services have stopped, click the Start All button.

2. Configure QAD Enterprise Applications

Completing the configuration of DDP requires the setup of the source application so that it connects to QXO and delivers the data that QAD QXtend expects. Several configuration steps are required for QAD EE.

2.1 Set Application Instance Name

When QAD EE directly publishes to QXO it needs to know where to deliver the data. This identity/name must match one of the source applications configured in QXO.

Set the Instance Name

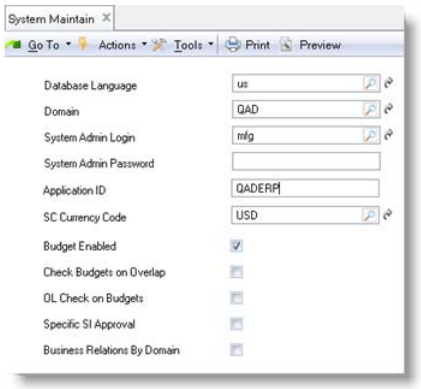
1 Open the QAD Enterprise Applications from the Windows Desktop.



2 Log in with User ID = `qmi`, password = “” (blank).

3 Open the System Maintain (36.24.3.1) function.

4 Set the Application ID to `QADERP` to match the source application name in QXO.

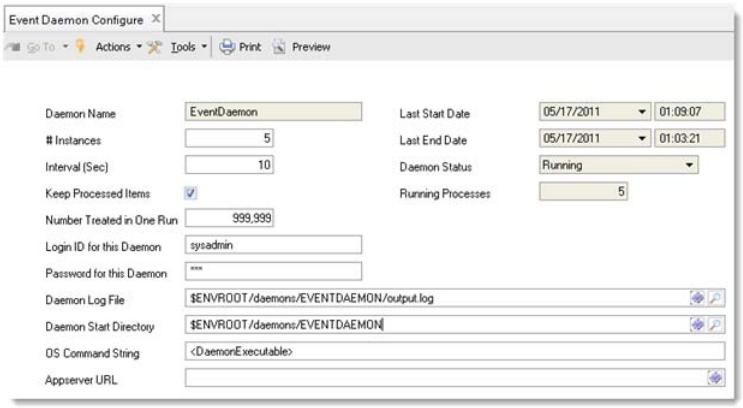


2.2 Verify the Event Daemon Configuration

The event daemon in the Financials module is used to publish data from business events and is required to publish data to QXO. Verify that it has been configured.

1 Open the Event Daemon Configure function.

2 Verify the configuration and the location of the log files.



2.3 Create Event Destination

The event daemon delivers business events to destinations that are configured in QAD EE. The destination identifies where the data should be delivered, and the way the data should be delivered. When configuring QAD EE to send data to QXO, you have to create a destination that sends data the QXO AppServer.

1 Open the Event Destination Create function.

2 Create the Event Destination with the following values:

   a Destination Name = `QXO DDP`

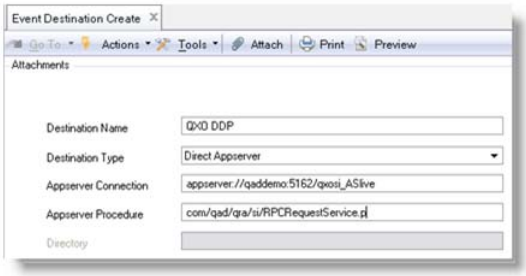
   b Destination Type = `Direct Appserver`

c Appserver Connection = `appserver://qaddemo:5162/live_qxo_AS`

This AppServer has been configured with the QAD QXtend Service Interface startup procedure enabled and is ready to receive requests.

d Appserver Procedure = `com/qad/qra/si/RPCRequestService.p`

3 Click the Save and Create button.



2.4 Create Event Configuration

The final step in the configuration is to identify the components that you want to publish data from QAD EE to QXO. Once you have the list of components, you create an event configuration record for each component. The event configuration record links the component events to a specific event destination, in this case QXO.

1 Open the Event Configuration Create function.

2 Create the event configuration with the following values:

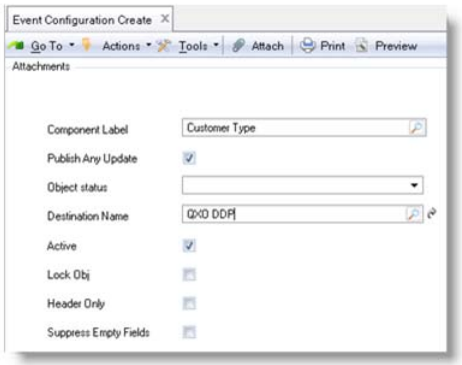
a Component = `Customer Type` (this is the DebtorType component).

b Select the Publish Any Update check box.

c Destination Name = `QXO DDP`

d Select the Active check box.

3 Click the Save and Create button.



Create another event configuration for the Journal Entry component.

1 Open the Event Configuration Create function.

2 Create the event configuration with the following values:

   a Component = `Journal Entry`

   b Select the Publish Any Update check box.

   c Destination Name = `QXO DDP`

   d Select the Active check box.

3 Click the Save and Create button.

3. Publish Data from QAD EE to QXO

You have now configured QXO and the data source QAD EE. Now we will process some transactions in the Financials module to test that the data is published to QXO and processed correctly.

3.1 Start Event Daemon

The event daemon is a background process that processes events that are raised by the application; this must be running for data to be published.

1 Launch the Event Daemon Start function.

2 You will get a message confirming that it has started.

3 To check that it is running, open the Event Daemon Monitor function. The daemon status should be running.

3.2 Create Debtor Type

1 Open the Customer Type Create function.

2 Enter Code = `DT01` and Description = `Debtor Type Test 01`.

3 Click the Save and Create button.

4 Verify that the message has been delivered to QXO:

   a Click the Subscriber Messages link on the Viewers tab.

   b Verify that the data in the message matches what was entered.

   c Review the data that is exported from QAD EE.

3.3 Create a Journal Entry

The Financials module in QAD EE contains some complicated business components and the APIs contain many fields. When first attempting to process data into QAD EE via QXI, you should create the transaction manually and publish the data to QXO where you can review what data needs to go into which fields. This will help you to understand what and where data has to go in the XML when loading it through QXI.

The Journal Entry is a fairly complex component. Once the data has been entered, it is published to QAD QXtend and there it can be reviewed and analyzed.

1. Open the Journal Entry Create function.
2. Enter the following details for the header: Daybook Code = `JE` and Description = `QXtend Training`.
3. Enter the following details for the first line: GL Account = `1100` and TC Debit = `100`.
4. Enter the following details for the second line: GL Account = `1090`, TC Credit = `100`, Allocation = New Open Item, Allocation Key = test.
5. Click the Save and Create button.

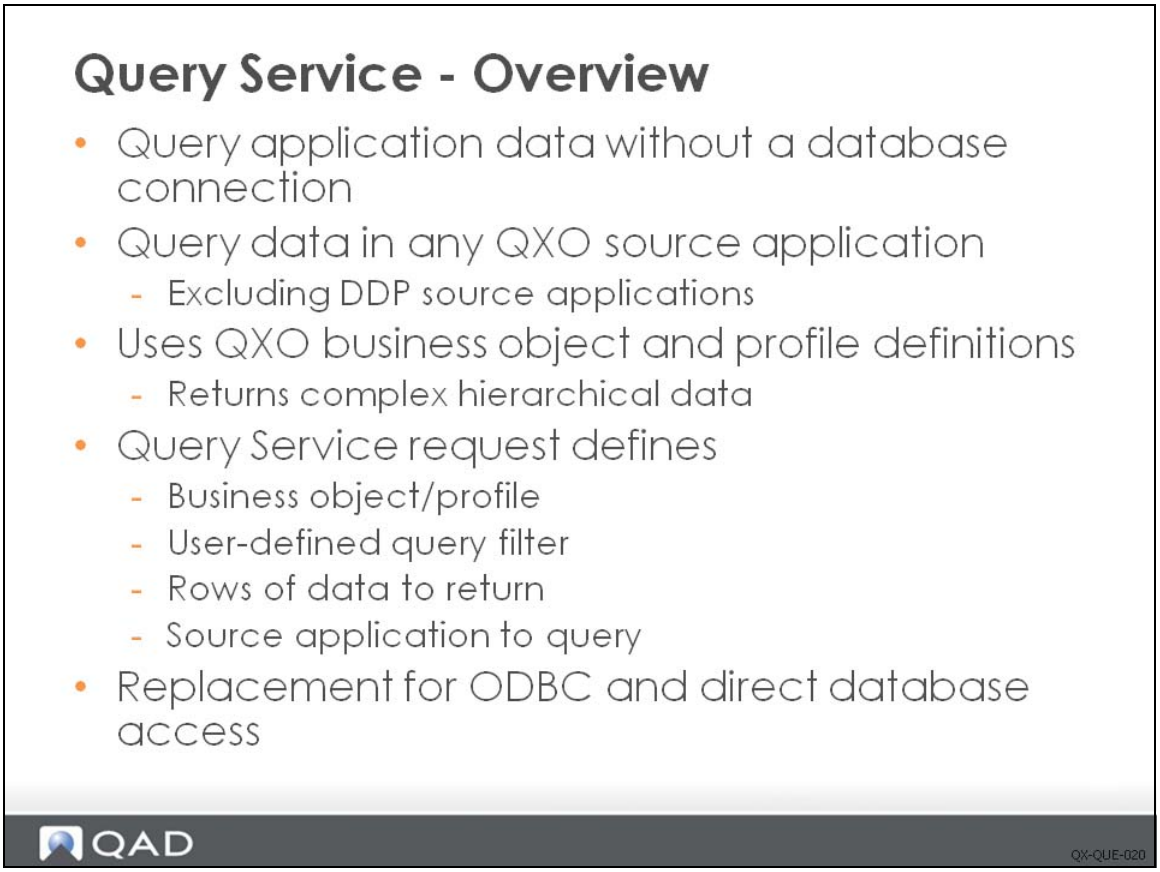
Verify that the message has been delivered to QXO by using the Subscriber Messages view on the Viewers tab. Verify that the data in the message matches what was entered.

Review the data that is exported from QAD EE.

Chapter 10

Query Service

Overview



The Query Service is an extension of the event extraction process used to extract and publish business event data from QAD applications based on rules defined in the QXtend configuration. The Query Service enables you to access data in QAD EA without needing direct access to the application database. Often, integration to external systems requires access to data in the QAD Enterprise Application, and this has typically been achieved with ODBC or direct database access. The Query Service allows you to replace those approaches with a standard QAD-supported solution.

The Query Service can query data in any source application configured in QXO; the only exception is that QXtend cannot query data from DDP source applications as they do not define application database sets. This means that using a single service in QXtend you can query application data from multiple data sources by changing a simple parameter in the Query Service request.

The data retrieved by the Query Service is not limited to a single table or a flat data structure; it will return a complex hierarchical data set either as a Progress ProDataset or XML document. The structure and format of the data returned is controlled by the business object and profile definitions configured within QXO. The business object definition is used to build and execute all of the queries required to fill the dataset from the source application database. The queries are executed using standard Progress 4GL constructs that provide superior performance over other methods, such as ODBC. The profile definition is used to map the data in the business object to the format defined in the profile, and this is used as the response to the Query Service request.

Requests made to the Query Service must provide the data required to perform the query. The request interface for the Query Service is generic and is the same regardless of the object being queried. However, the data that is returned will obviously be different. The request must supply the following information:

Profile. Name of the profile in QXO you are querying; this is linked directly to the business object. This lets you choose which view of the data from the business object you would like returned. If the profile includes calculated fields and fixed values, these also are returned in the response. If the default profile name is supplied (that is, the same name as the business object), the data is returned in the format defined in the business object.

Query Filter. A filter can be defined that is applied to the query that fills the top level of the business object; the filter allows you to limit the data that is returned in the response. For example, you could return information for a single sales order by filtering on the sales order number.

Note The filter can only be against data in the data source for the top level table. That is, if the top level is `so_mstr` you can only filter on data from `so_mstr`; you cannot include other tables in the filter.

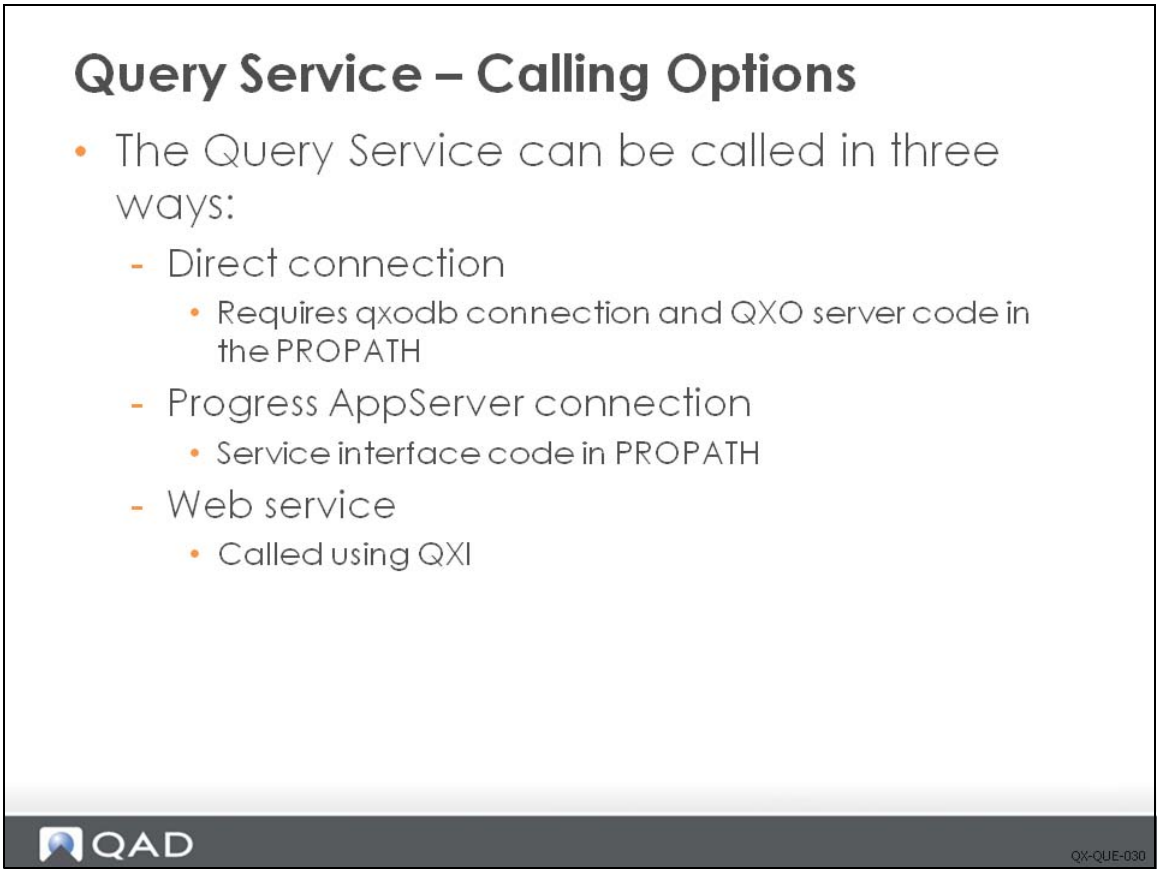
Source Application. Name of the source application instance configured in QXtend to query. This must be a valid QXO source application that does not use DDP.

Max Rows. Number of rows in the top level table that should be returned if the query returns more than one row. This allows you to limit the size of the dataset that gets returned.

Ignore BO Filter. Whether ignore BO filter. If this is set to true, then BO filter wonít be used when extracting data

Ignore BO Inner Join. Whether ignore BO inner join. If this is set to true, then BO inner join wonít be used when extracting data.

Calling Options

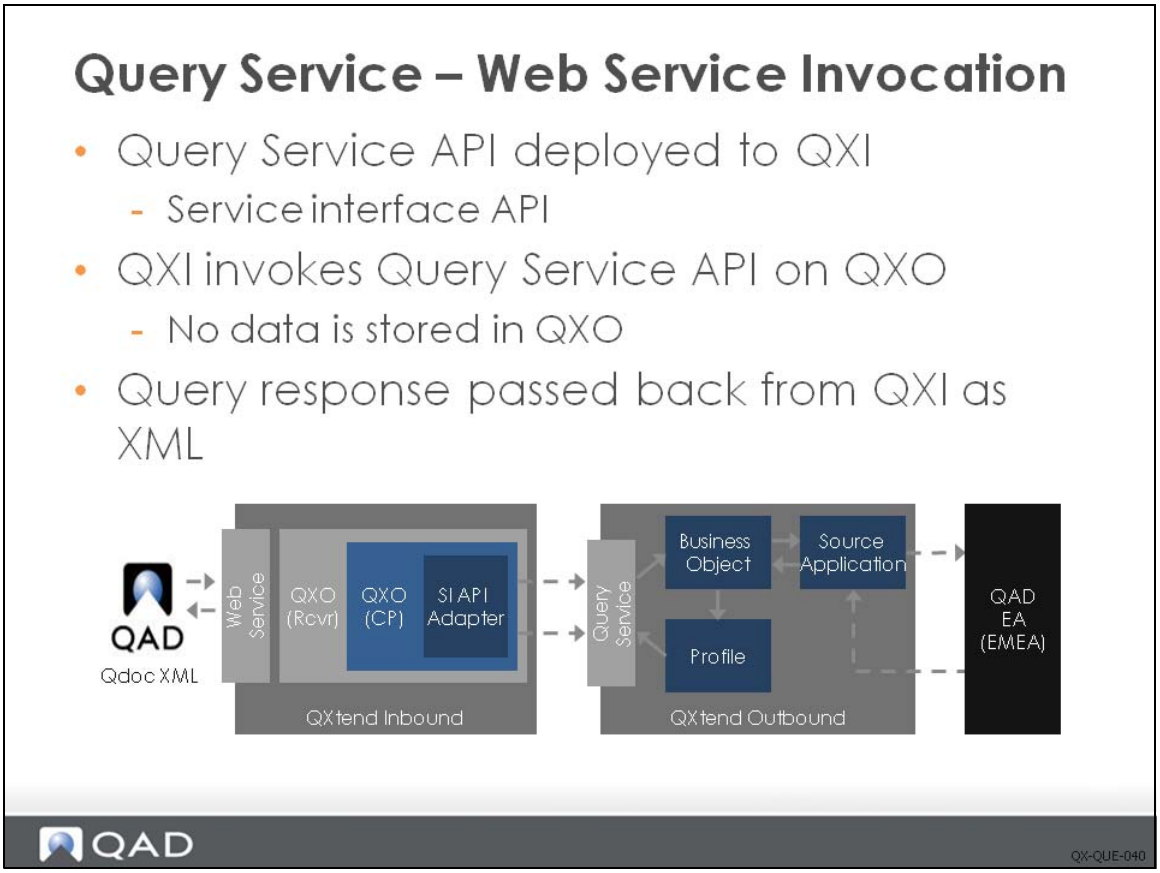


The Query Service is implemented using the SI layer, so you have several supported methods when it comes to using it from your other application:

- Direct connection: Requires an Progress OpenEdge (10.1B+) session that connects to the QXO database and then executes the actual query API. This method also requires that you have all of the QXO server runtime code in the session PROPATH. This method executes the Query Service API without going through the SI. The query result is returned as an instance of a ProDataset.
- AppServer connection: Requires that the client making the request is one of the clients supported by the AppServer (AppServer supports OpenEdge, .NET, and Java clients). The client connects to the AppServer and invokes the API, using the SI to process the request. The response data for the query is returned as a ProDataset.
- Web service: This method can be used by any application that can process an `http` message. This method can easily be integrated into an Enterprise Application Integration (EAI) or Enterprise Service Bus (ESB) architecture, as Web services fit naturally into this space. The Web service is hosted as a normal service in QXI, which connects to QXO via an AppServer and invokes the API on QXO. In this method, response data is returned to the caller as an XML document.

The following section discusses using the Web Service method for calling the Query Service. See *Technical Reference: QAD QXtend* for details on the other two invocation methods.

Web Service Invocation

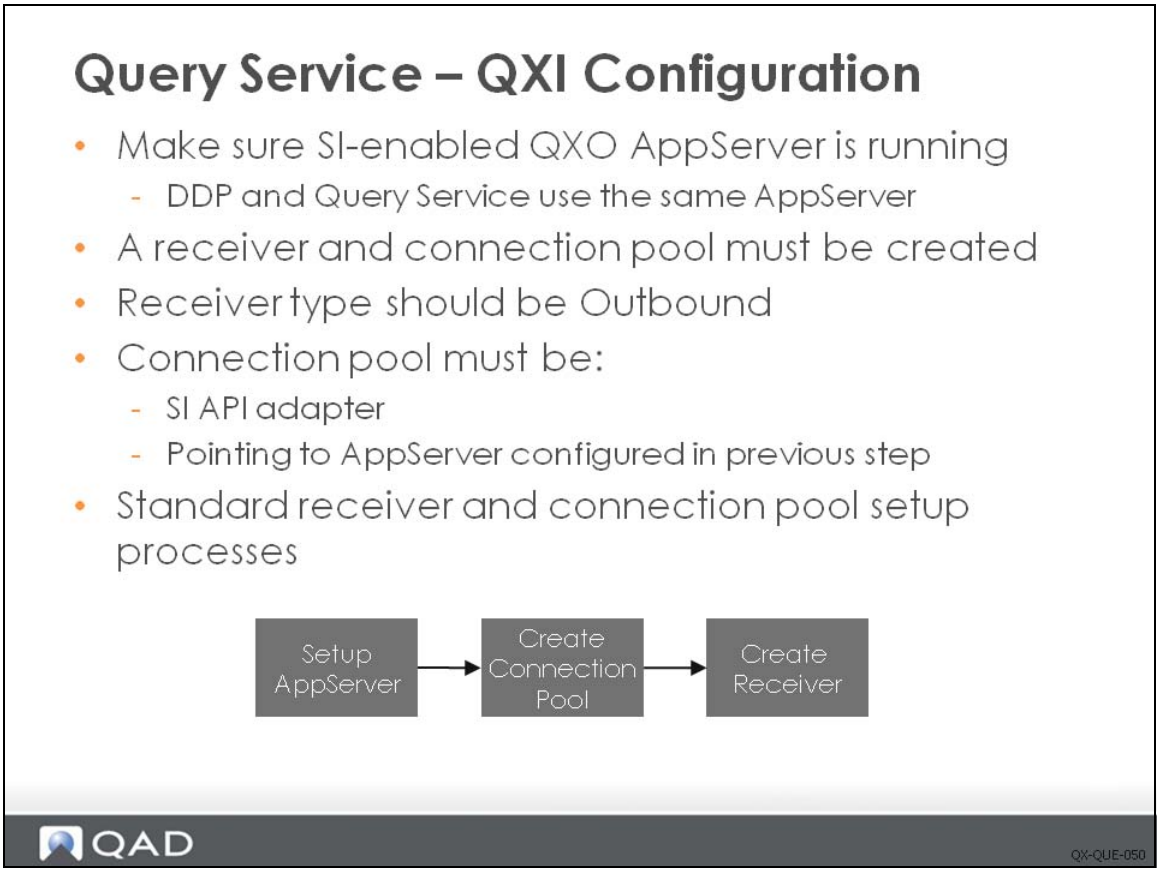


The Query Service API generated from the QXO profile screen is deployed to QXI using Deploy Query function. The API is deployed as an SI API.

When a Query Service request is deployed to QXI, it must be addressed to a receiver that has a connection pool that is connected to the QXO SI-enabled AppServer. The request details are passed from QXI to QXO, where the query request is processed. The query is executed against the relevant source application that is identified in the request, and the result of the query is returned to QXI.

The result of the query is packaged as the response to the QXI request and passed back to the caller as a SOAP XML message.

QXI Configuration

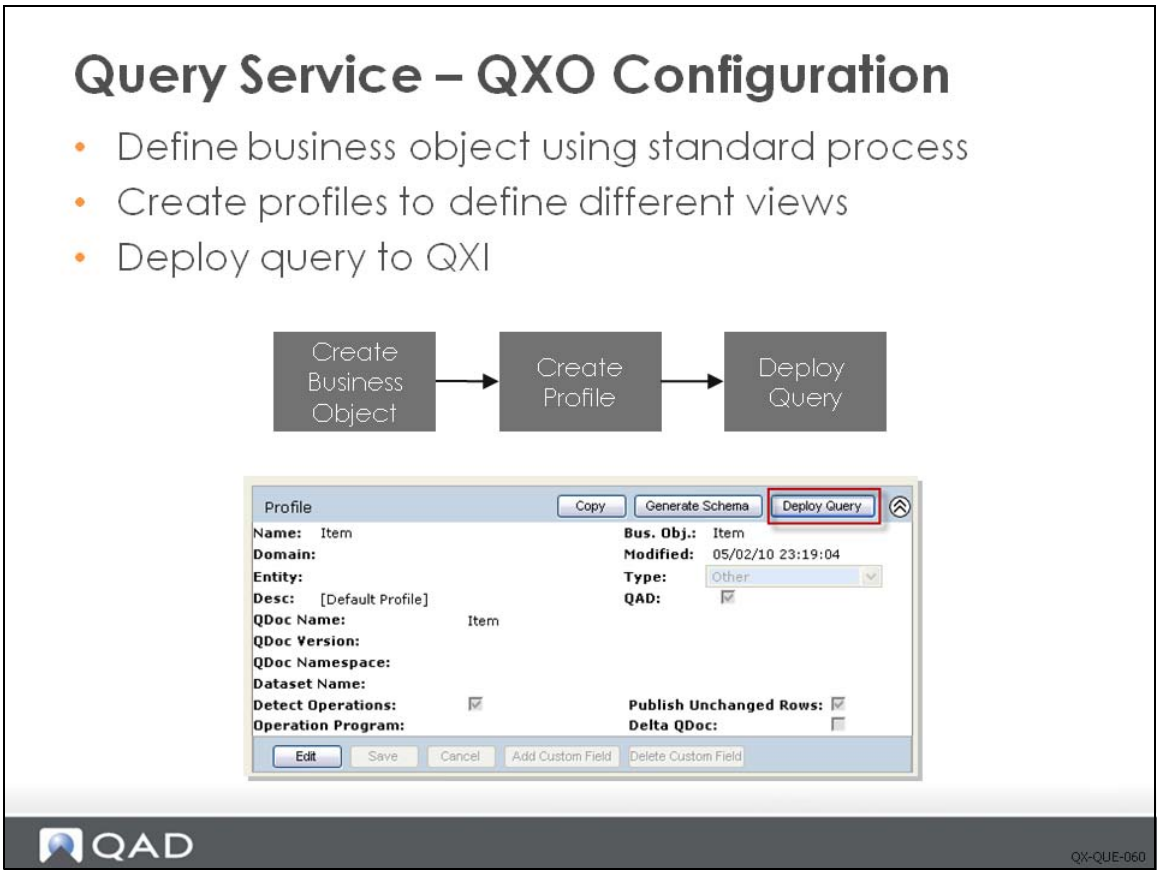


The Query Service runs against QXO. In order for it to be called from QXI, an SI-enabled AppServer is required. The AppServer instance must be connected to the QXO database. The QXO runtime code is required in the PROPATH, and it must use the QXO SI startup procedure `com/qad/qxtend/si/AppServerStart.p` when it starts.

By default such an AppServer has been configured for DDP and Query Service during installation of Qxtend. The AppServer is shared by DDP and Query Service.

Follow the standard configuration process for QXI. Any request to be processed by QXI requires a receiver and connection pool. The receiver type of the receiver must be Outbound. The connection pool created must be a SI API adapter connection pool. It must be configured to work with the AppServer created for the Query Service requests.

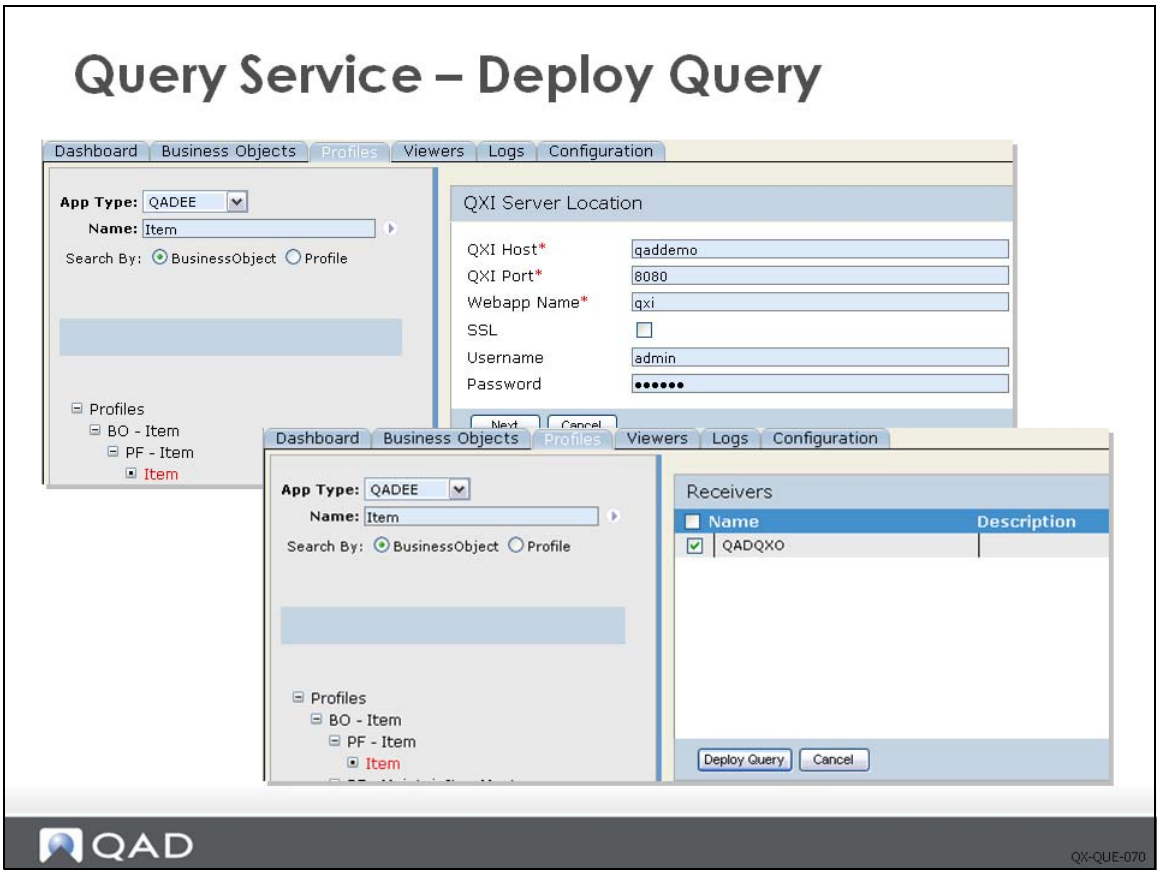
For details on creating an SI API connection pool, see Chapter 8, “Service Interface Layer,”in the training guide.



The first step in using the Query Service is to define the structure of the response data from a Query Service request. After determining the data you want returned, create a business object definition that contains all the data you identified. Before creating a new business object, verify that none of the existing objects contains the data you need. When you create your business object, make sure that the business object only contains the fields you need.

The business object defines the raw message structure. The structure probably will not meet your requirements since you have no control over the field names within the returned data; you also may need additional calculated fields that are not stored in the database returned from the query. Create or use an existing profile to ensure that the data returned from the Query Service meets the requirements of the system that is querying the data.

After defining the business object and profile you need for your query, you can automatically deploy the query API to QXI by using the Deploy Query function.



To deploy the query service API to QXI, QXO must establish a connection to the QXI server:

1 Provide the host and port information for the QXI server. You also need to specify the Webapp name.

2 If using Secure Socket Protocol https encryption, select the SSL check box.

Note If SSL is selected, then QXI Host must include domain name of the host.

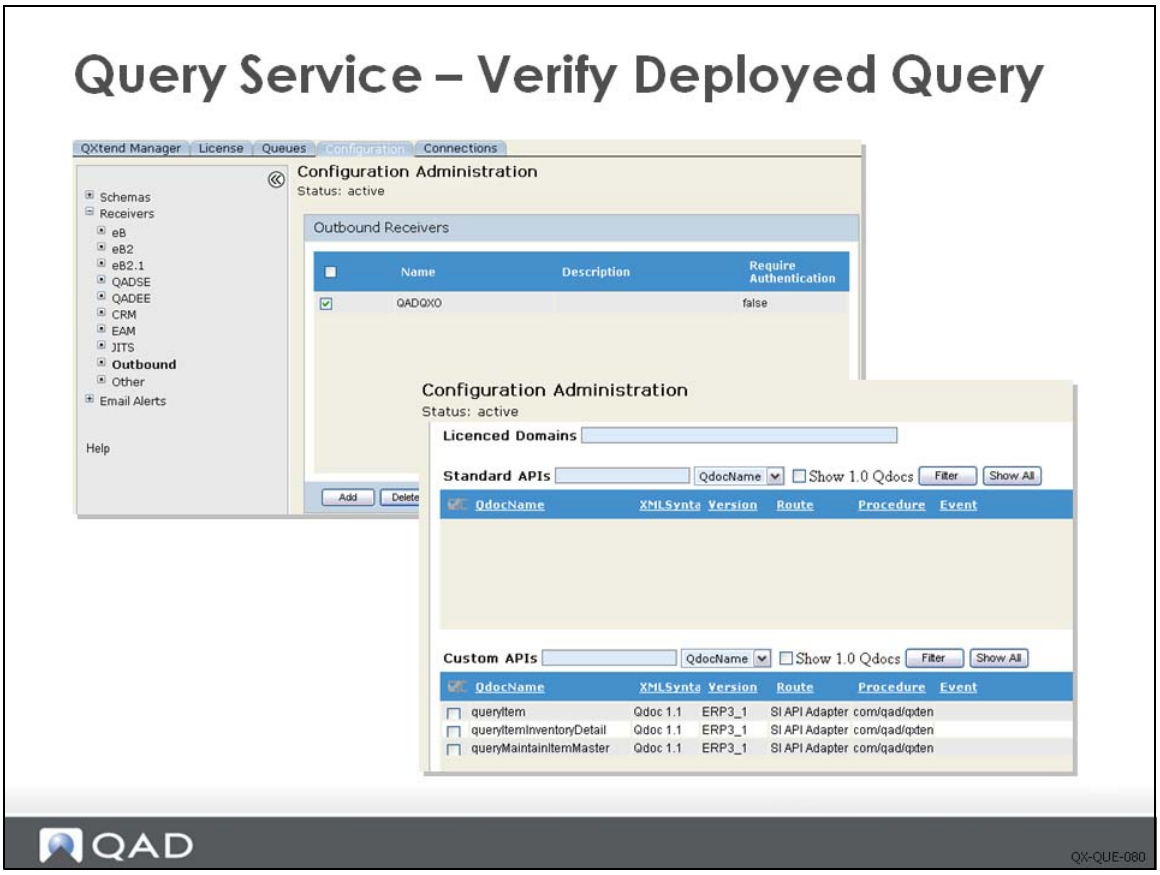
3 Enter user login credentials and then click Next.

Note The system assumes the same user login credentials as for QXtend Outbound.

Once the connection to QXI has been established, QXO will display the receivers for the Outbound module in QXI.

4 Select the receiver or receivers to which you want to deploy the query API, and then click Deploy.

5 A system message displays the status of the deployment.



To verify that a query API has been deployed successfully, you can go to QXI and view the schemas under the receiver for query service. A new custom API should be identified. Typically, it has following attributes

QdocName: The name of the Qdoc is ‘query’ + profile name. For example, if the profile name is ‘Item’, the QdocName of the API will be queryItem.

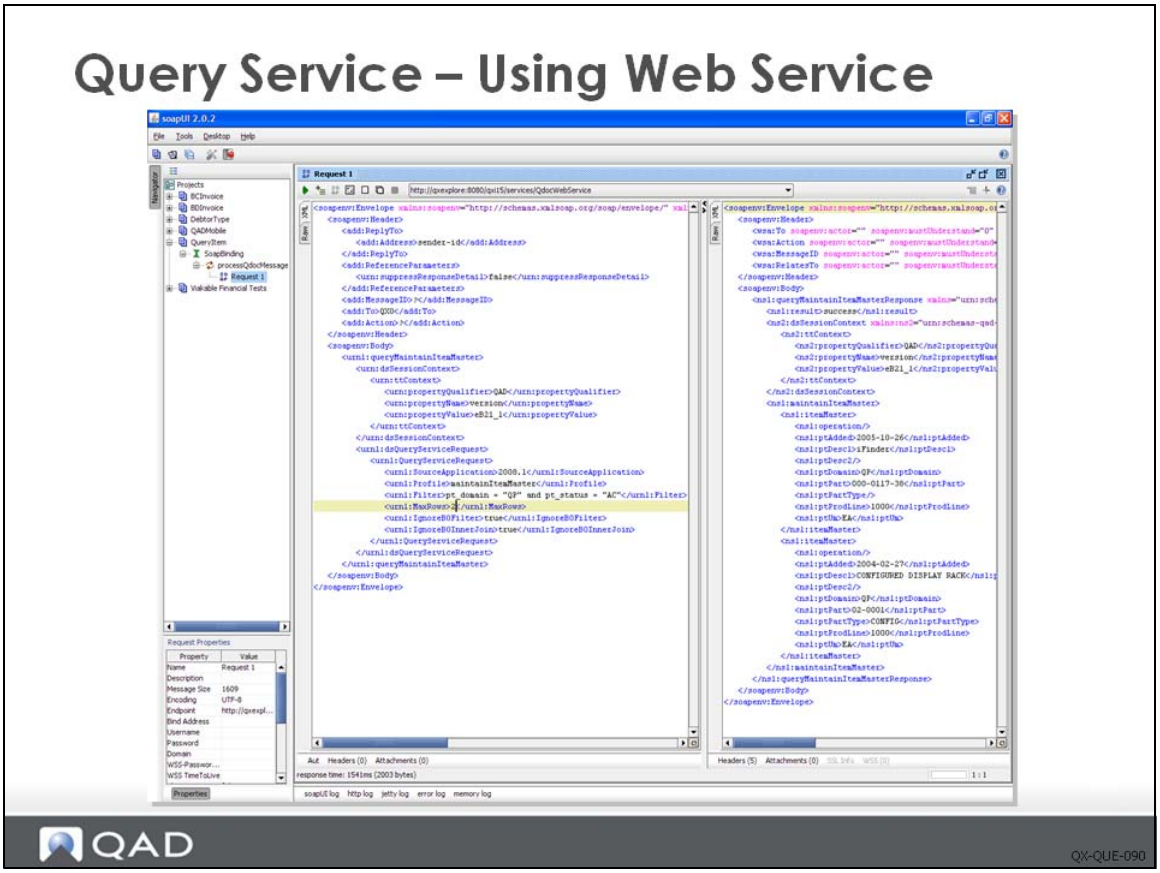
XML Syntax: Qdoc 1.1.

Version: Depends on the type of source application. For example, if source application type is QADEE, the version of the query API will be ERP3_1.

Route: SI API

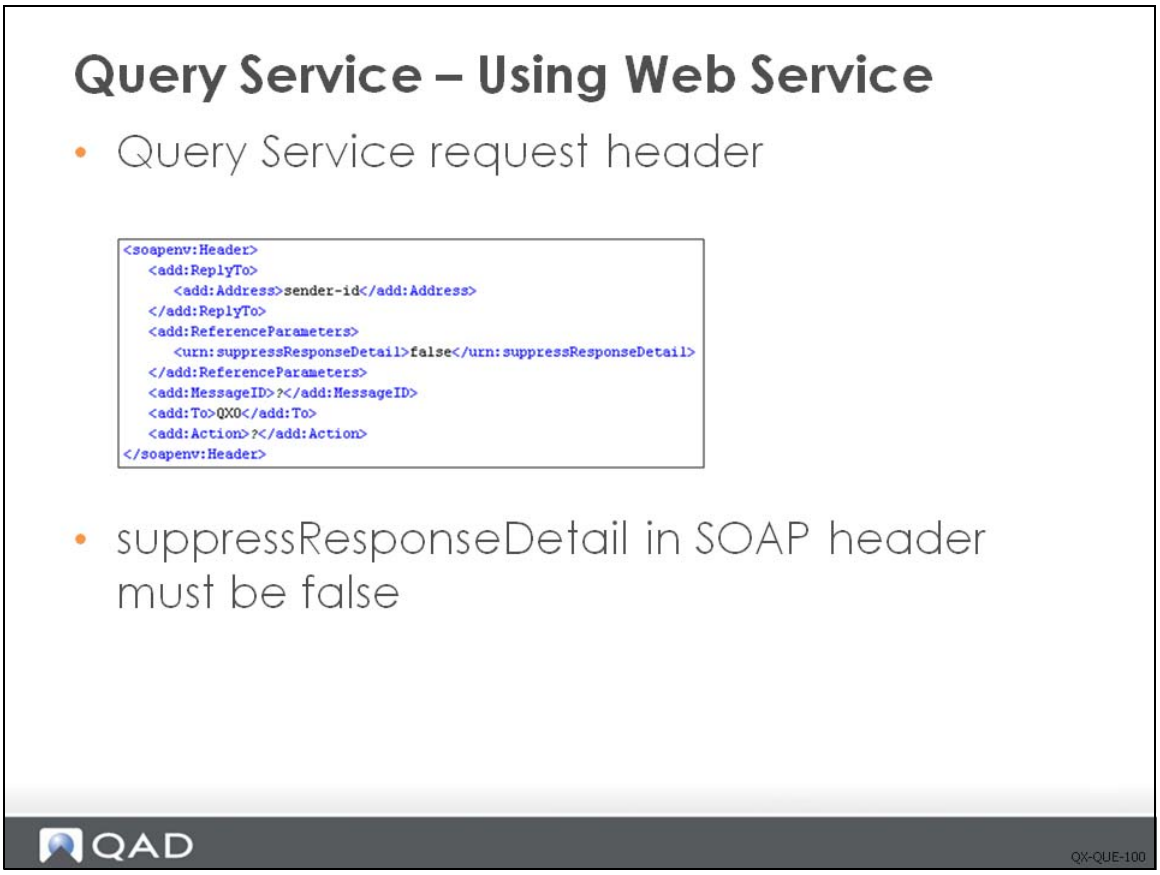
Procedure: `com/qad/qxtend/si/QueryService.p`

Using Web Service



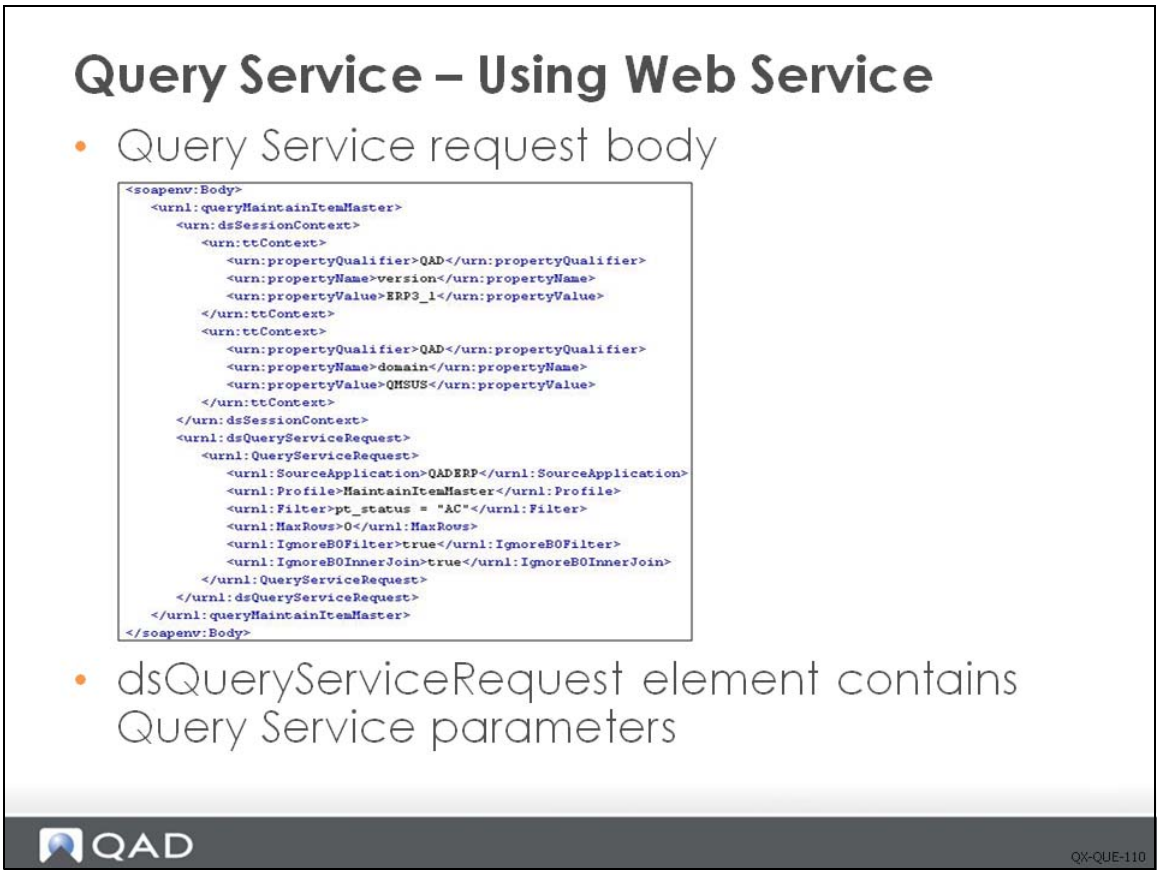
The simplest way to test the Query Service functionality once it has been deployed is to use the SOAP UI toolset to build some test cases. The Generate WSDL button in QXI can be used to generate WSDL and schemas which are used to generate the new WSDL project in SOAP UI.

Once a new project has been built in SOAP UI, new requests can be created and executed against the APIs deployed in QXI. For details on using SOAP UI, see the section "*Testing QXI with SOAP UI*" in the training guide.



The header parameters in the SOAP envelope are identical to all other QXI requests. However, the `suppressResponseDetail` parameter must be set to false.

The `suppressResponseDetail` parameter controls whether or not details are returned on the response. If the parameter is set to "`true`" only the request processing status and exceptions are returned, so data returned from the Query Service will not be part of the response message from QXI.



The body part of the SOAP message is where the actual request data that is passed to the Query Service is populated.

Session context parameters of Query Service engine:

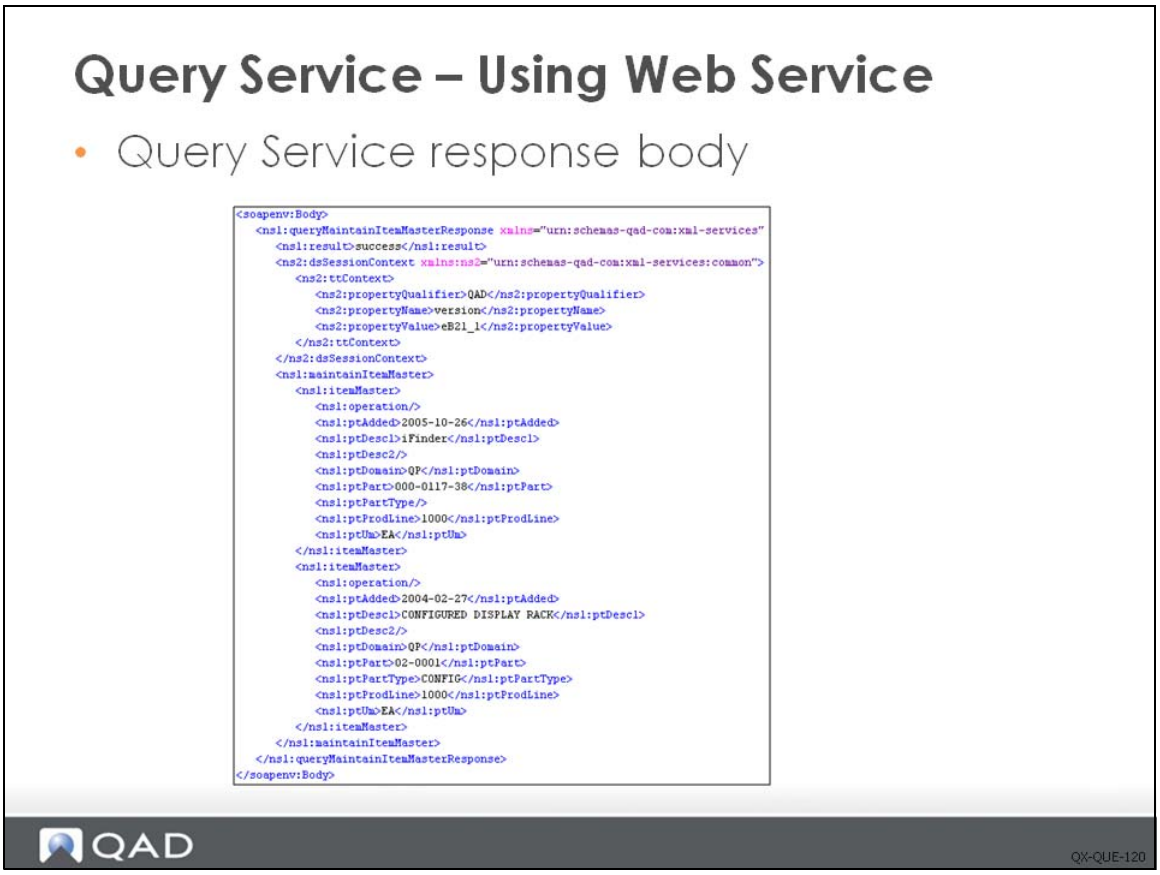
- `Version:` This context entry is mandatory since all requests to QXI use this to determine the version of the API to use when processing the request.
- `Domain:` The QXO query service only allows querying from one domain at a time. If a business object is domain-enabled, the domain must be passed in the request; otherwise, an error is returned.

The data that drives the Query Service is provided in the `dsQueryServiceRequest` element; this applies for any Query Service regardless of the business object being queried. The Query Service is a generic routine and exposes a standard interface for other applications to leverage. The advantage of this is that external applications can build a Query Service caller mechanism to execute all queries if necessary, since the interface stays the same and only the data changes.

Six parameters drive the Query Service engine:

- `SourceApplication:` Name of the source application instance in QXO that you want to query data from. This identifies the data source for the query. The name must not point to a DDP source application.
- `Profile:` The name of the profile in QXO. This identifies the business object being queried and the profile used to control the format used to return the query result.

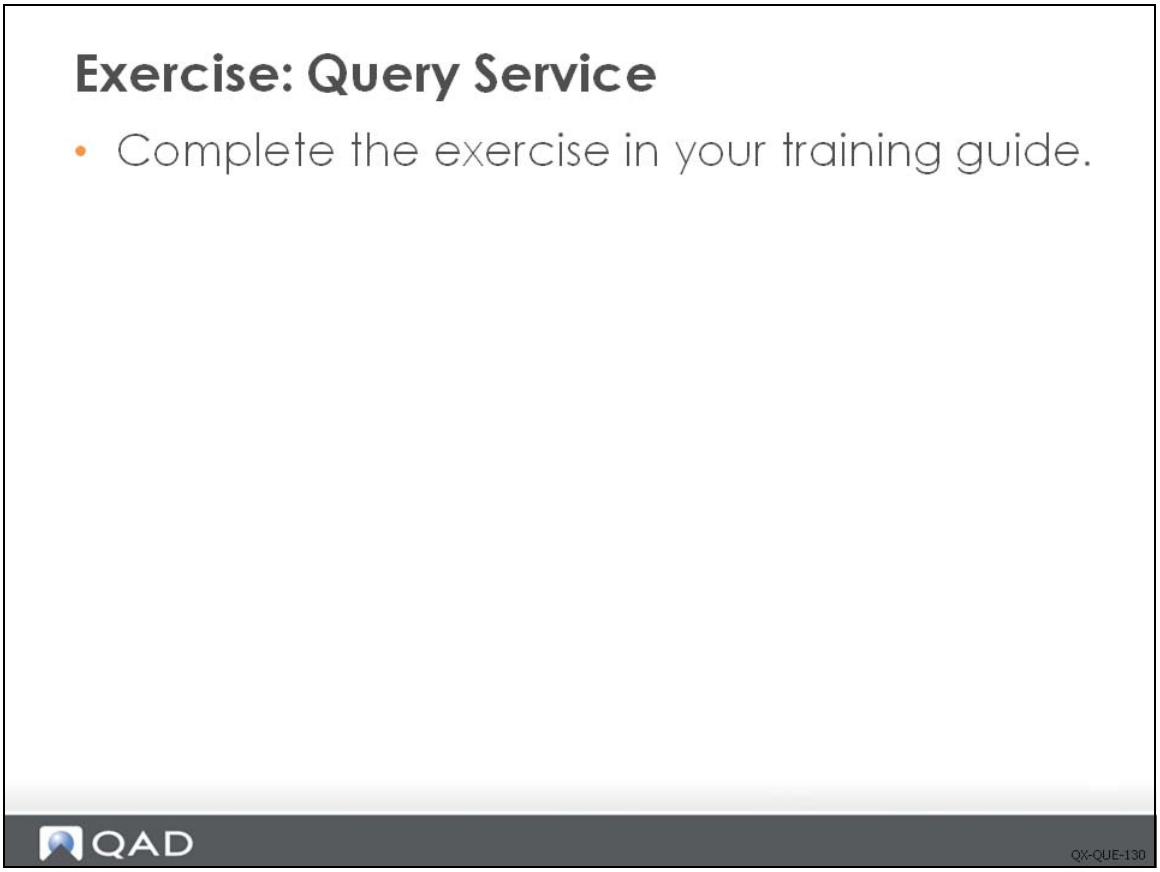
- `Filter:` Valid OpenEdge `where` clause without the `where` statement. This filters the data returned by the query. The filter only restricts the data that is returned in the top-level table of the business object; it does not affect the child records that are returned. The filter can only be against data in the data source for the top-level table in the business object. That is, if the top level is `so_mstr`, you can only filter on data from `so_mstr`; you cannot include other child tables in the filter.
- `MaxRows:` Controls the maximum number of rows returned for the top-level buffer in the business object. If the business object is defined with child tables, all child records are returned for all records in the top buffer. That is, if a sales order query is limited to a max of ten rows, ten sales orders will be extracted. If each order has ten lines, all 100 lines also will be returned. Setting `MaxRows` to 0 returns all data that matches the filter criteria.
- `IgnoreBOFilter:` When set to true, ignores any filters defined against the top-level table in the business object definition.
- `IgnoreBOInner Join:` When set to true, ignore any inner join defined against the business object.



The data that is returned from the Query Service is returned from QXI as the Web service response message. The response data is part of the response SOAP body in an element called `query<ProfileName>Response` where *`ProfileName`* is the name of the profile defined in the Query Service request message.

The format and contents of the Query Service response is controlled by the profile definition in QXO. The response is returned as a hierarchical XML document that visually shows all parent-child relationships in the returned data.

Exercise: Query Service



The following list shows a number of key concepts used in the Query Service in QAD QXtend. In each statement below, fill in the correct term from the list.

native Progress call	generate WSDL
Profile	deploy query
single-character expression	external application
ProDataSet	Source Application

1. The QAD QXtend Query Service is a QXO service that allows an _____ to access data stored in a QAD application.
2. The Query Service connects to the source application database, extracts and packages the data, and returns it to the calling application as XML or as a _____.
3. The Query Service can be invoked in any of three ways: _____ (direct connection), Progress AppServer call, or XML Web Service call.
4. A calling application can specify filter conditions to control the data returned by the Query Service. These are defined as a _____ or string.
5. To use Query Service through third party application like SOAPUI, you must first _____ from QXO to QXI and then _____ for the API in QXI.
6. The _____ and _____ in Query Service request QDocs are used to specify the data source.

Lab: Query Service

QAD QXtend 1.7.2 has been installed with both QXI and QXO components. In the previous labs you performed the required tasks to allow QAD QXtend to work with the Financials module in QAD Enterprise Edition. The Query Service allows you to query application data from external applications using QXO business objects and profiles. This lab describes how to use the Query Service from external applications.

`<LabHomeDirectory> = C:\QXtendTraining\Labs\10-QueryService\`

External non-QAD applications that need to integrate with QAD EA often need to access QAD Enterprise Applications data. Typically this is done via ODBC connections to the database or native database connections. These connections can be troublesome, especially ODBC. The Query Service can be used to provide more effective access to QAD EA data.

1. QXO Configuration

Several steps must be followed to configure QXO.

1.1 Configure QXO Service Interface AppServer

Accessing the Query Service from an external application requires a Progress AppServer that is used to host the Query Service API. The AppServer must have the Service Interface enabled.

Verify that there is a QXO AppServer available that is Service Interface enabled.

1. Open PUTTY by using the icon on the training Windows machine Desktop.
2. Double-click the `qaddemo` saved session.
3. Log in as user `demo-admin`, password `qad`.
4. View the following file:
   `/dr01/progress/dlc/properties/ubroker.properties`
5. Search for `com/qad/qxtend/si/AppServerStart.r`
6. Note the AppServer name.

1.2 Create Business Objects

The queries that are executed by the Query Service are controlled by the business object definition in QXO. You need to build a business object that contains all the data you want to query. Remember that a business object has multiple levels. The business object definition gives you complete control over the data returned.

1.2.1 Item Inventory Business Object

In this scenario we have an external application that needs to access information about an item and the inventory levels of that item across the system. Use the following information to build the business object:

BO Name = Item Inventory

Top Level Table = pt_mstr

Fields

- oid_pt_mstr
- pt_desc1
- pt_desc2
- pt_domain
- pt_part
- pt_part_type
- pt_price
- pt_status
- pt_um

Child Table = in_mstr

Parent table = pt_mstr

Join = pt_domain,in_domain,pt_part,in_part

Fields

- in_domain
- in_site
- in_loc
- in_loc_type
- in_part
- in_qty_all
- in_qty_avail
- in_qty_oh
- in_qty_ord
- oid_in_mstr

Validate the business object and ensure that it validates without any errors or warnings.

1.3 Create Profile

1. Copy the current profile and call it `ItemInventoryDetail`.
2. Set the QDoc name to `ItemInventoryDetail`.

Now map the business object created in Step 1.2 using the following details:

1. Rename `pt_mstr` data object to `item`.
2. Rename the `pt_mstr` fields to:
   - `pt_part` to `item`
   - `pt_part_type` to `itemType`

- pt_desc1 to primaryDescription
- pt_desc2 to secondaryDescription
- pt_um to unitOfMeasure
- pt_status to itemStatus
- pt_price to itemPrice

3 Exclude oid_pt_mstr

4 Rename the in_mstr data object to itemInventory.

5 Rename the in_mstr fields to:

- in_part to item
- in_site to site
- in_loc to itemLocation
- in_loc_type to locationType
- in_qty_all to quantityAllocated
- in_qty_avail to quantityAvailable
- in_qty_oh to quantityOnHand
- in_qty_ord to quantityOrdered

6 Exclude oid_in_mstr

2. Query Service – Call from QXI

You can call the Query Service in three ways:

- QXI Web Service
- Progress AppServer
- Direct Progress connection

In this lab you will access the Query Service using the QXI Web Service method. For details on the other methods, refer to *Technical Reference: QAD QXtend*.

2.1 Deploy Query to QAD QXtend Inbound

2.1.1 Create New Receiver in QXI

The receiver that you have used in previous labs has been communicating with the QAD EE instance, but for the Query Service you need to direct the requests to the QXO instance. You need to create a new receiver and a new SIAPI connection pool. The Connection Pool connects to the QXO AppServer instance that was identified in Step 1.1 and should be an SI-enabled AppServer.

2.1.1.1 Create Receiver

To create a new receiver:

1 Open the QXI Web application in Internet Explorer: http://qaddemo:8080/qxi

2 Select the Configuration tab.

3 Select the Receivers menu option.

4 Select Outbound from the Receivers menu.

5 Click the Add button.

6 Choose the Continue Configuration update without suspending QXtend Inbound option and click the Submit button.

7 Set the Receiver Name to `QADQXO` and leave the Licensed Domains field empty.

8 Continue to the API Selection screen.

9 Do not select any APIs; these will be added later.

10 Complete the transaction by clicking the Done button.

The receiver has been created but it is not yet ready to use; you first have to define a way to connect to the target QXO instance.

2.1.1.2 Create Connection Pool

To create a connection pool:

1 Open the QXI Web application in Internet Explorer `http://qaddemo:8080/qxi`

2 Select the Connections tab.

3 Expand the Add Connection Pool menu.

4 Select the Add SIAPI Pool option.

5 Set the following values:

- Pool Name = `QADQXO` (must be identical to the receiver name created in the previous step)
- App Server Name = `qxosi_ASlive`
- Host = `qaddemo`
- Port = `5162`
- User = `qmi`
- Password = “” (blank)
- Minimum Connections = `1`
- Maximum Connections = `2`

6 Click Save to create the new connection pool.

Now check that you have configured it correctly and have idle sessions available:

1 Open the QXI Web application in Internet Explorer: `http://qaddemo:8080/qxi`

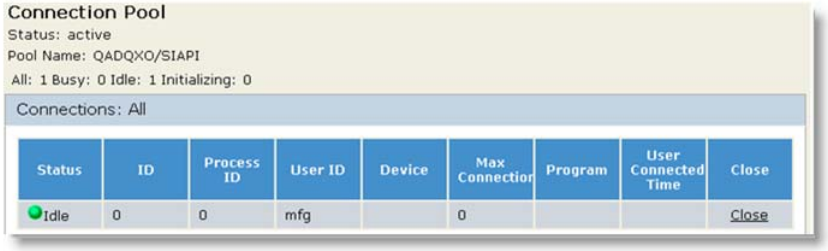
2 Select the Connections tab.

3 Expand the View Connection Pools menu.

4 Select the QADQXO.SIAPI option from the menu.

5 Expand the Connections menu.

6 You should see the connection pool with a single idle connection as below. If there is not an idle session, the connection pool has not been configured correctly.



2.1.2 Deploy Query API to QXI

Deploy the new Query API schema and WSDL to QXI as a custom schema.

1 Open the QXO Web application in Internet Explorer: `http://qaddemo:8080/qxo`

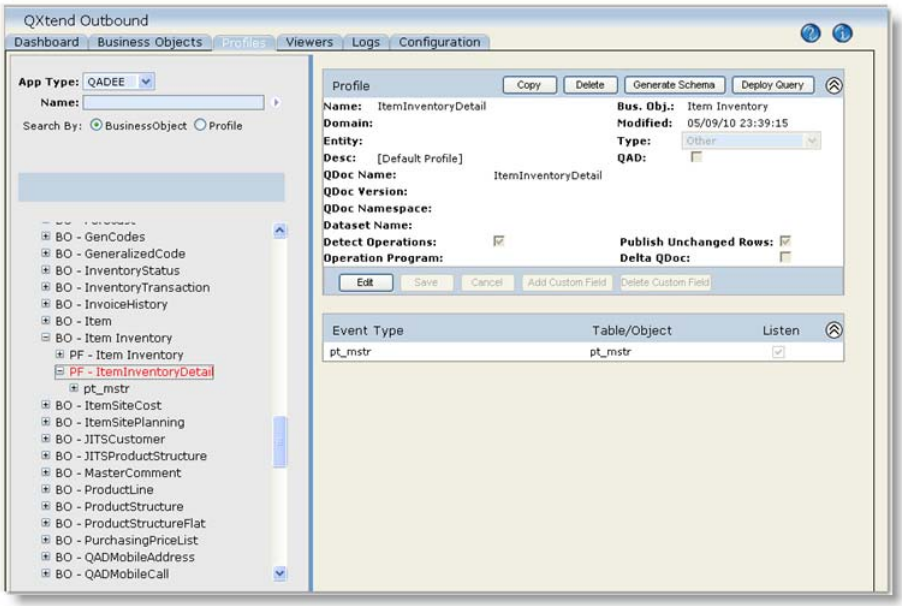
2 Select the Profiles tab.

3 Ensure that the App Type is set to QADEE.

4 Click the fetch button to get all profiles.

5 Select the `ItemInventoryDetail` profile under the Item Inventory business object.

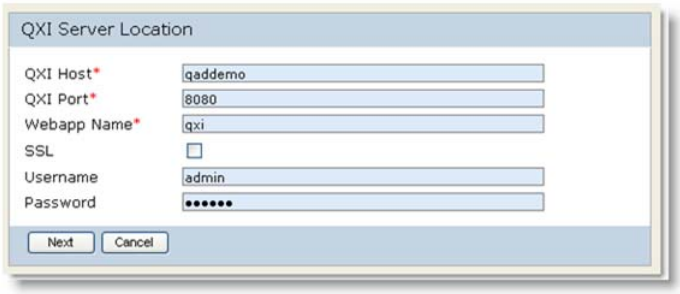
6 Click the Deploy Query button.



7 Set the following values:

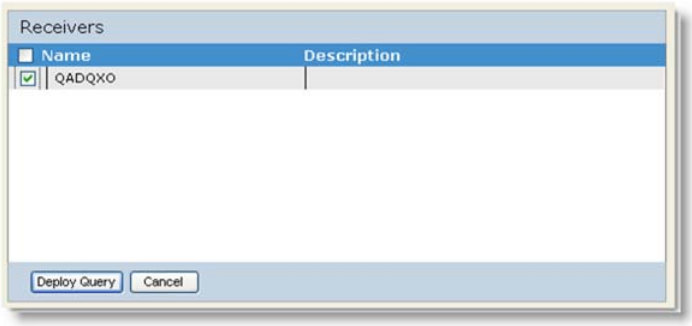
QXI Host	qaddemo
QXI Port	8080
Webapp Name	qxi
SSL	no

Username	admin
Password	mfgpro



8 Click Next button.

9 Choose QADQXO and click Deploy Query button.



2.1.3 Verify the queryItemInventoryDetail API has been added

To verify that the queryItemInventoryDetail API was successfully added to QXI, you need to check two places. First check that the API is available as a custom API for the Outbound application type; second, check it is a supported API on the QADQXO receiver.

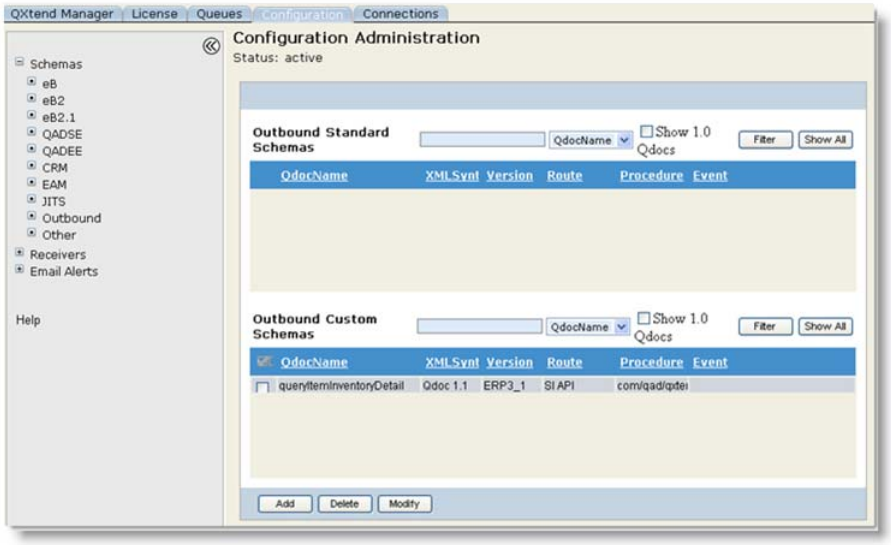
1 Open the QXI Web application in Internet Explorer: `http://qaddemo:8080/qxi`

2 Select the Configuration tab.

3 Select the Schemas node on the menu tree.

4 Select Outbound.

The queryItemInventoryDetail version ERP3_1 should display as the custom API that is available for the Outbound application type as shown here:



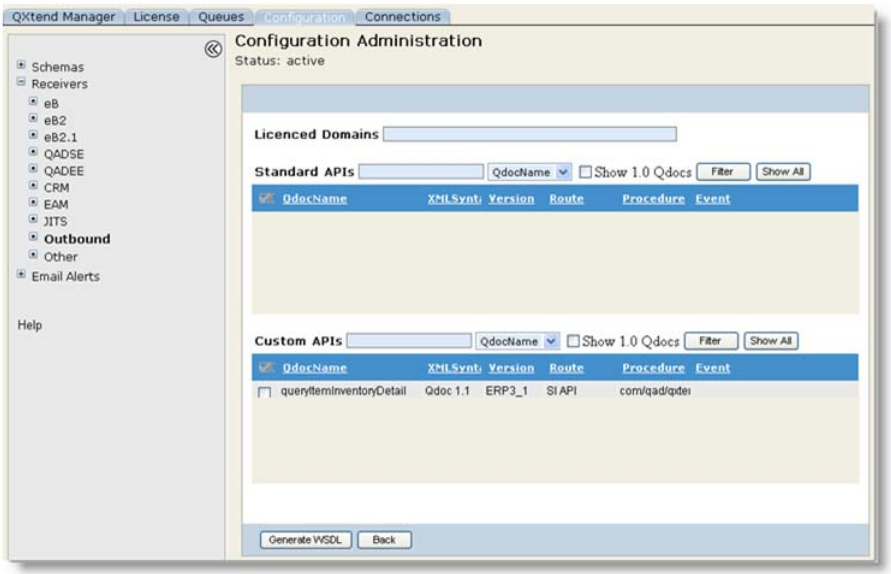
5 Select the Configuration tab.

6 Select the Receivers node on the menu tree.

7 Select Outbound.

8 Select the check box next to the QADQXO receiver and click the View button.

The queryItemInventoryDetail API version ERP3_1 should display as a supported QDoc for the receiver as shown here:



2.2 Test the Query API

The QueryItemInventoryDetail API has now been successfully deployed to QXI. However, until you use the new version to process a couple of transactions you cannot be sure that everything has been uploaded correctly. Use soapUI to build a WSDL project, create some requests, and process those requests using soapUI.

Generate the WSDL file first:

1. Open the QXI Web application in Internet Explorer: `http://qaddemo:8080/qxi`
2. Select the Configuration tab.
3. Select the Receivers node on the menu tree.
4. Select Outbound from the list of receivers.
5. Select the check box next to the QADQXO receiver and click the View button.
6. Select queryItemInventoryDetail from the list of Custom APIS.
7. Click the Generate WSDL button.
8. Click the No button for “Create backward compatible WSDL?”
9. Save queryItemInventoryDetail-ERP3_1.zip to `<LabHomeDirectory>`
10. Unzip the file to `<LabHomeDirectory>/queryItemInventoryDetail-ERP3_1`.
11. Open soapUI on the Windows image. Use the shortcut on the Desktop.
12. Create a new workspace:
    - File – New Workspace
    - Set the workspace name to `Lab 10 – Query Service.`
    - Save the workspace file in `<LabHomeDirectory>`.
13. Create a new WSDL Project.
    - Right-click the workspace name.
    - Select the New soapUI Project option.
    - Set the project name to Query Service 10.
    - Set the WSDL file to `<LabHomeDirectory>/queryItemInventoryDetail-ERP3_1/queryItemInventoryDetail-ERP3_1.wsdl`
    - Click OK to create the project.
14. Drill down to the Request 1 message created under the Query Service 10 project.
15. Right-click Request 1 and select the Clone Request option. Enter a name for the new request.
16. Edit the new request so that it can be processed by QXI. Change SOAP header:
    - Set the receiver to `QADQXO`.
    - Set `suppressResponseDetail` to `false` (this must be `false` otherwise the result of the query will not be returned).
17. Create the necessary session context entries. Create one `ttContext` iteration:

- Qualifier = `QAD`, Name = `version`, Value = `ERP3_1`
- Qualifier = `QAD`, Name = `domain`, Value = `10USA`

18 Edit the application data section of the message:

a The `dsQueryServiceRequest` node contains the fields available when processing a query.

b Edit the XML and use the following values:

- SourceApplication = `QADERP`
- Profile =`ItemInventoryDetail`
- Filter = “”(blank)
- MaxRows = `0`
- IgnoreBOFilter = `false`
- IgnoreBOInnerJoin = `false`

19 Process the QDoc message you created in the new soapUI project and review the response message. Examine the data that is returned.

Modify the filter condition to return different data:

1 Clone the request created in the previous step.

2 Set the filter to `pt_part_type = "COMP"`.

3 Process the request and review the response. The returned set of data is different.

Change the configuration of the Profile in QXO:

1 Modify the profile in QXO so that the `itemInventory` data object is not included in the profile.

2 Clone the request created in the previous step.

3 Process the request and review the response. The returned set of data is different.

Change the configuration of the profile in QXO:

1 Modify the profile in QXO:

a Set the `itemInventory` data object back to being included in the profile.

b Add a filter to the `itemInventory` data object so that only inventory records with a non-zero quantity on hand are returned in the query.

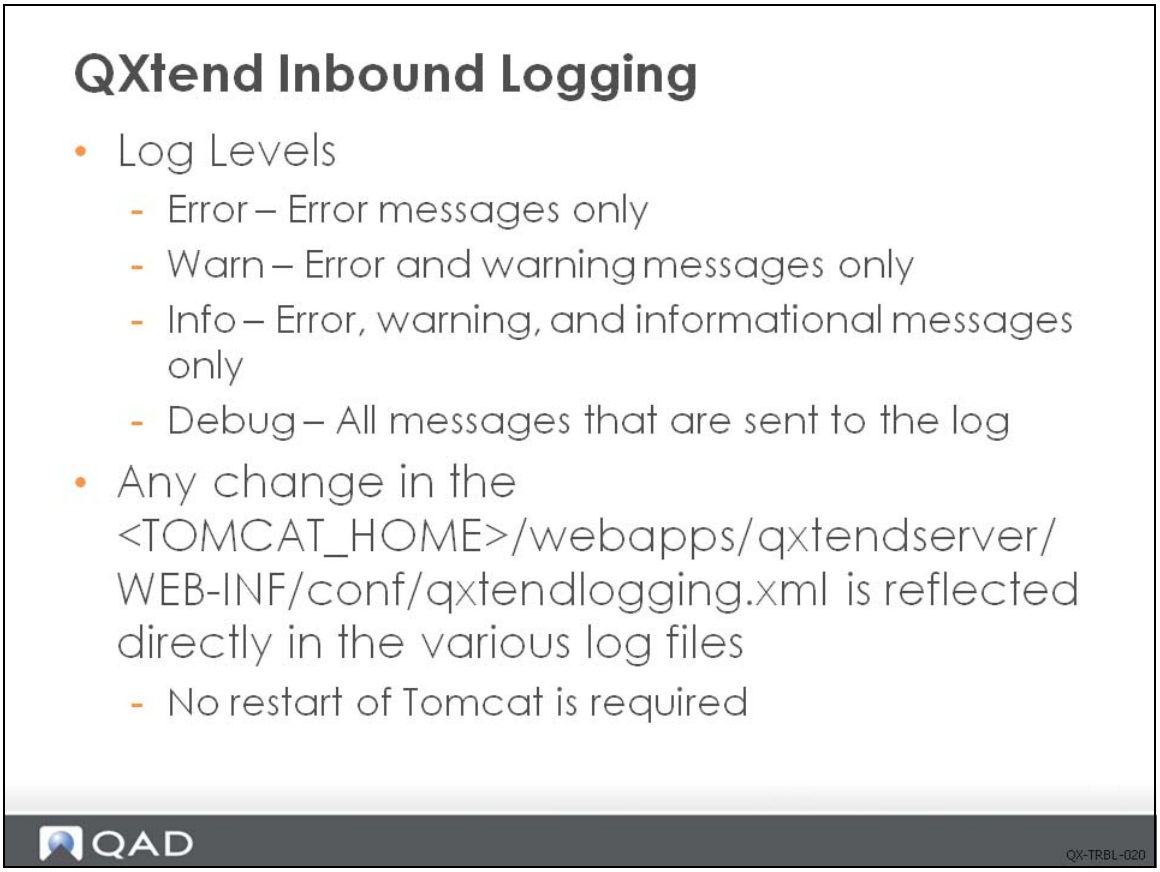
2 Process the request and review the response. The returned set of data is different.

3 Try changing the query to see what other filters can be applied using the Query Service.

Chapter 11

Diagnostics and Troubleshooting

QAD QXtend Inbound 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 QXI, 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 directed, do not change the details for these logs, except where they are stored or the reporting levels. You might want to move these files to backup storage or delete them when the records are no longer required.

Reporting levels for the QXI logs are:

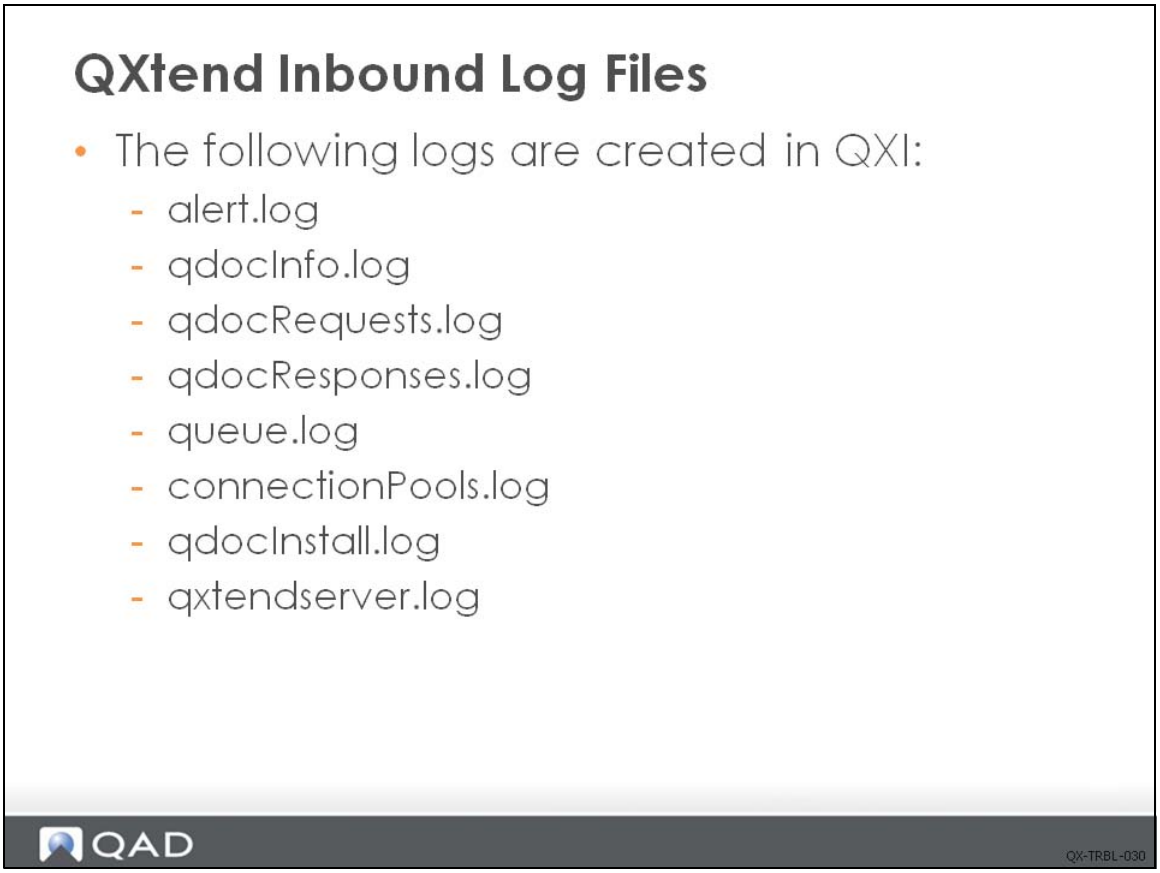
- Error: error messages only
- Warning: error and warning messages only
- Info: error, warning, and info messages only
- Debug: all messages

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

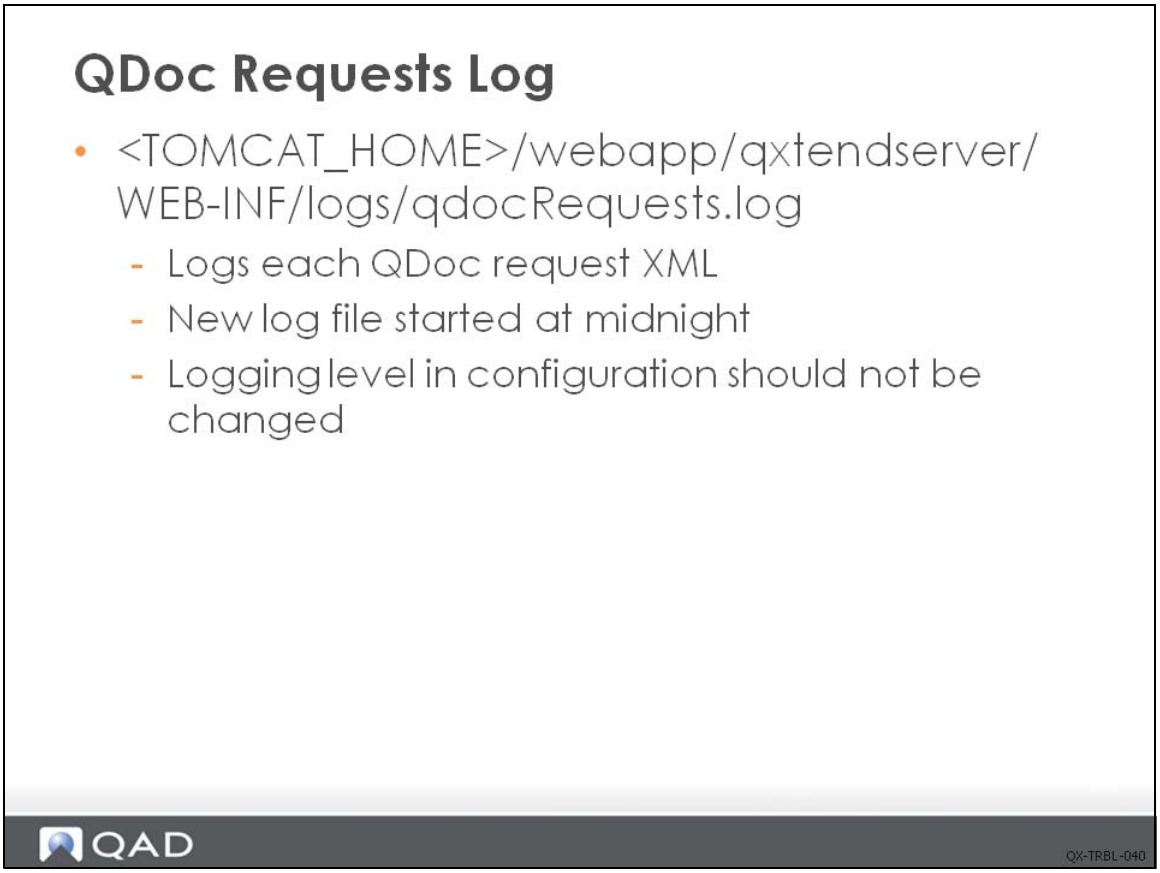
The logs created in QXI are:

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

QAD QXtend Inbound Log Files



QDoc Requests Log



Each QDoc Request that is received by QXtend is logged in the `qdocRequests.log` file. The date, time, and other information is recorded and the request XML is also recorded. The log file is created in:

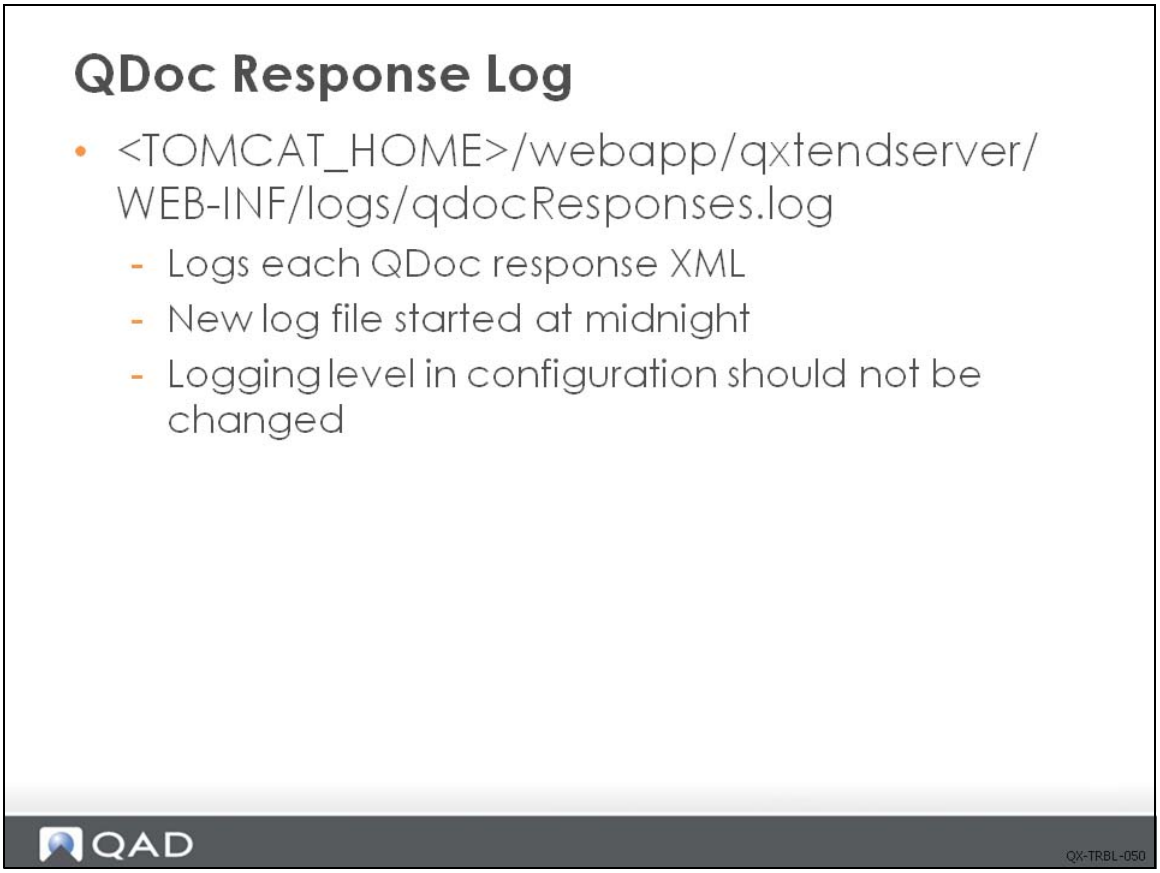
```
<Tomcat-dir>/webapps/qxtendserver/WEB-INF/logs
```

The log file is configured so that each day at midnight the log file rolls over and archives the existing log file with a date suffix. Then a new log file is started.

The logger node in `qxtendlogging.xml` is:

```
<logger name="qdocLogger.request" additivity="false">
     <level value="info"/>
     <appender-ref ref="asyncqdocRequest"/>
</logger>
```

QDoc Response Log



Each QDoc response that is returned by QXtend is logged in the `qdocResponses.log` file. The date, time, and other information is recorded along with the response XML. The log file is created in:

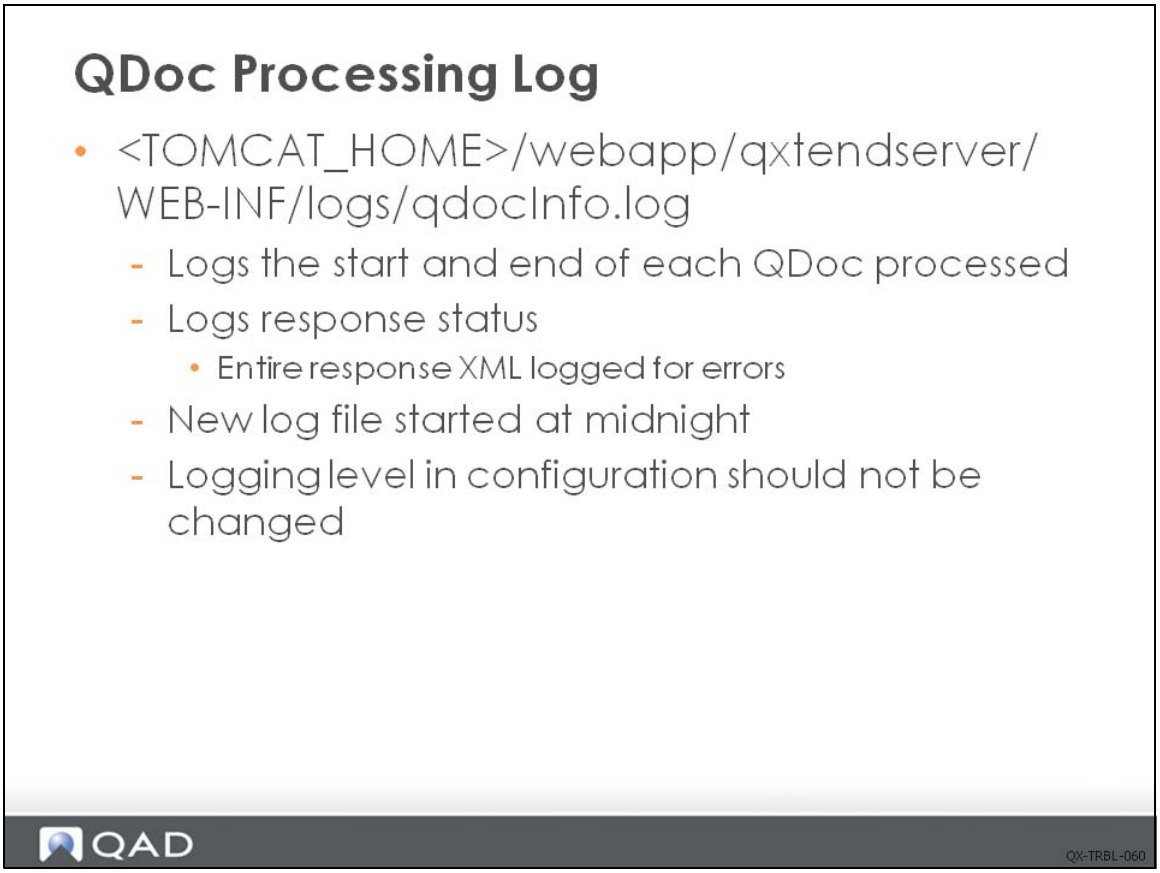
```
<Tomcat-dir>/webapps/qxtendserver/WEB-INF/logs
```

The log file is configured so that each day at midnight the log file rolls over and archives the existing log file with a date suffix. Then a new log file is started.

The logger node in qxtendlogging.xml is:

```
<logger name="qdocLogger.response" additivity="false">
     <level value="info"/>
     <appender-ref ref="asyncqdocResponse"/>
</logger>
```

QDoc Processing Log



Each request that is processed is logged in `qdocInfo.log`. A one-line entry is created indicating that the request had been received, a transaction number is assigned, and the transaction ID is retrieved from the request. The responses are also logged in this log file: the transaction number is logged as well as the transaction ID, along with a string that identifies the status of the processing (success/warning/error). If a response contains an error, the whole response XML is logged to this file.

Each request and corresponding response are logged in the `qdocInfo.log` file; the date, time, and other information is recorded. The log file is created in:

```
<Tomcat-dir>/webapps/qxtendserver/WEB-INF/logs
```

The log file is configured so that each day at midnight the log file rolls over and archives the existing log file with a date suffix. The a new log file is started.

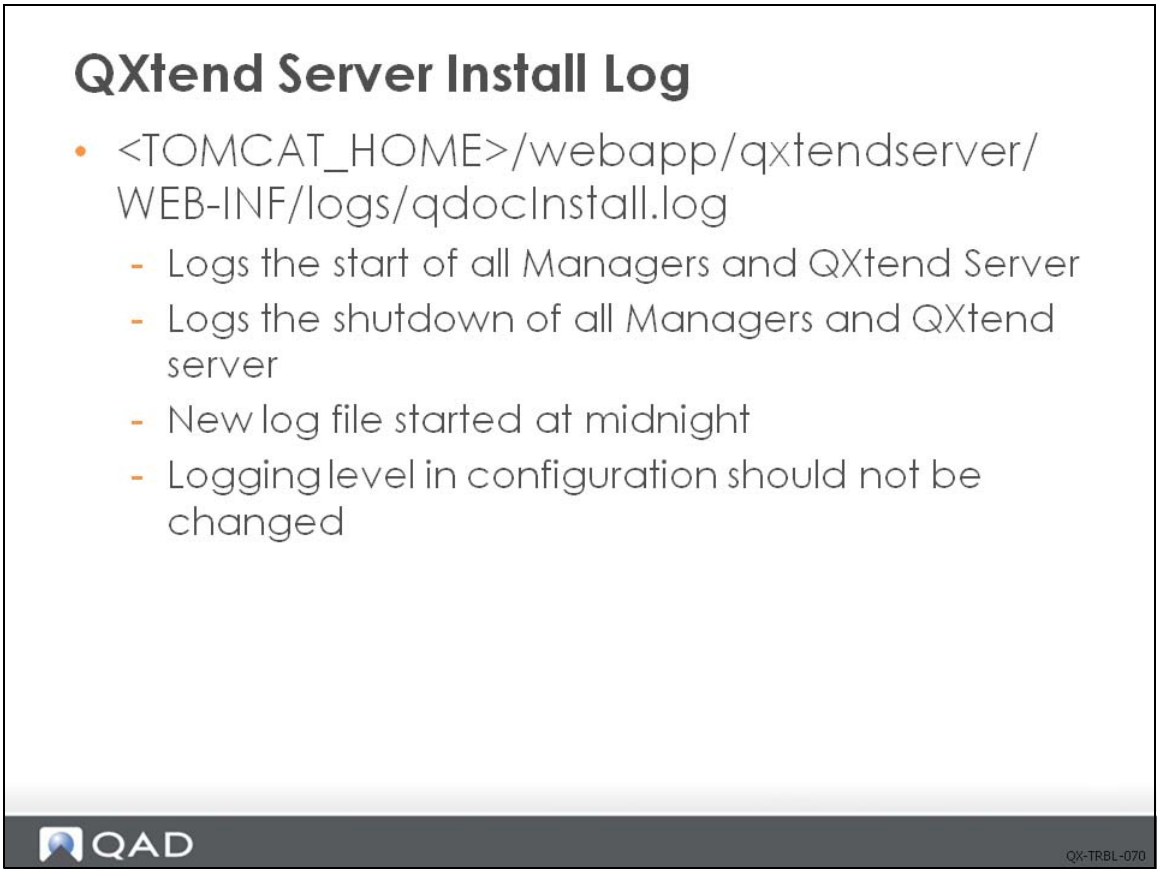
A sample non-error entry is:

```
2009-02-03 01:43:46,797 INFO qdocLogger.info [1233250081771][-
]API Request Received

2009-02-03 01:44:14,185 INFO qdocLogger.info [1233250081771][]API
Request Complete - success
```

The message line components are ISO format date (`2009-02-03`), ISO format time (`01:43:46,797`), reporting level (`INFO`), logger (`qdocLogger.info`), unique message ID (`1233250081771`), and message (`API Request Received`).

QAD QXtend Server Install Log



The `qdocInstall.log` file contains all details about the startup and shutdown of the Environment Manager and its Managers. This log is updated with details when the QXI Webapp is started, stopped, or reloading, listing components and whether they initiated or terminated successfully.

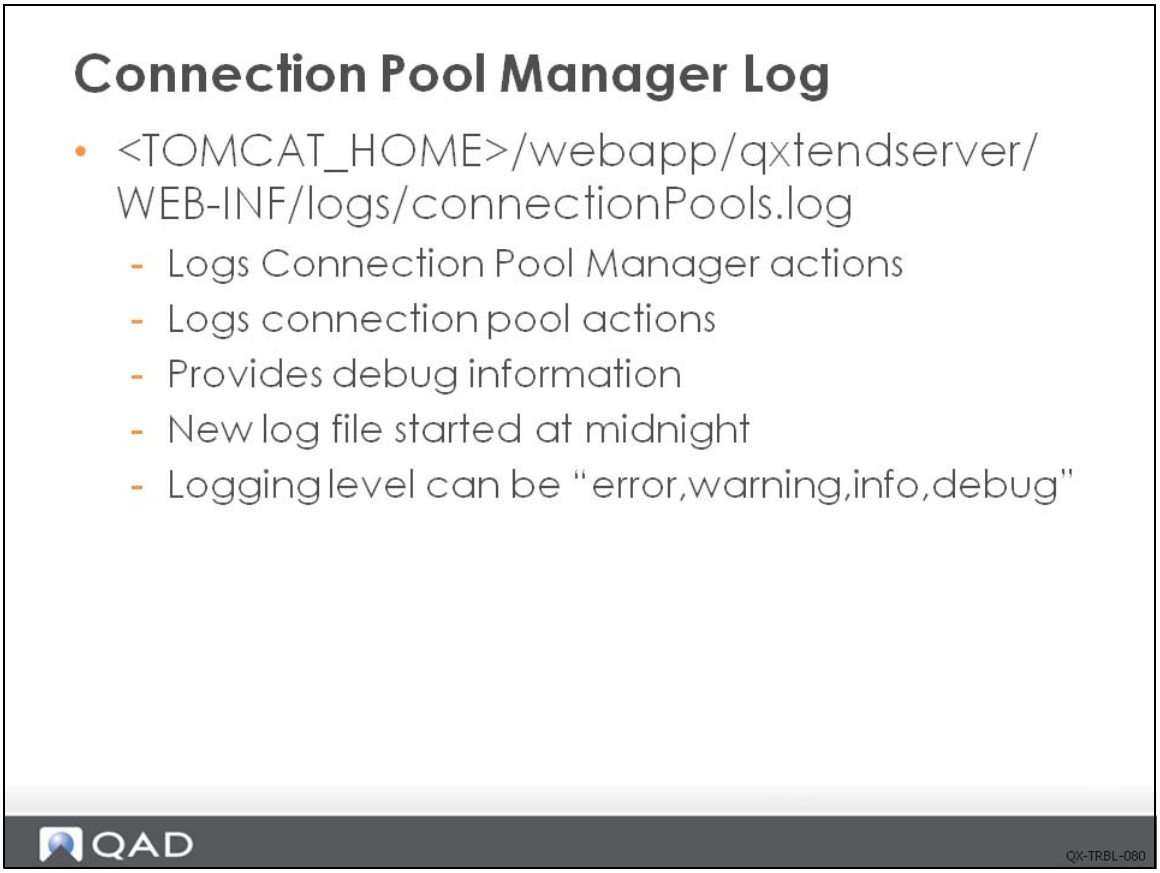
This log has a maximum of five files (current + four previous), with a sequence number. The file is rolled over based on log size, which is currently set to 500K; for example, `qdocInstall.log.1`. Once the maximum number of files has been created, QXI reuses the existing files. You do not need to delete these files since they always take up finite storage space.

The startup and shutdown details are logged in the `qdocInstall.log` file; date, time, and other information is recorded. The log file is created in:

`<Tomcat-dir>/webapps/qxtendserver/WEB-INF/logs`

The log file is configured so that each day at midnight the log file rolls over and archives the existing log file with a date suffix. Then a new log file starts.

Connection Pool Manager Log



Activity from the Connection Pool Manager and each of the connection pools that it manages is logged in this file. The default setting for the logging level of this log file is INFO, so only the information messages raised from the code are recorded. The logging level can be adjusted so that debug/warning information is also displayed to the log file. The logging level is changed in the `qxtendlogging.xml` configuration file.

The logger in qxtendlogging.xml is:

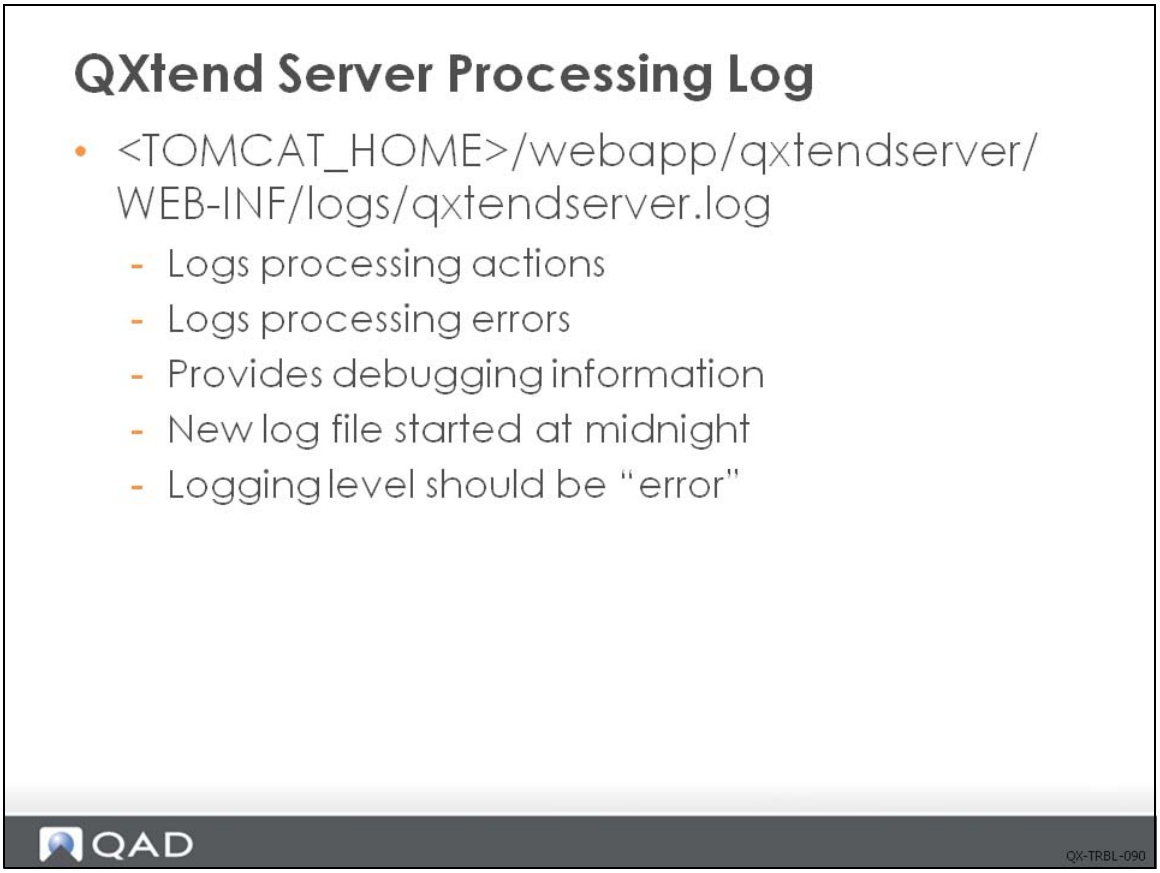
```
<logger name="connectionPool" additivity="false">
     <level value="info"/>
     <appender-ref ref="connectionPoolApp"/>
</logger>
```

All activity with the connection pools is logged in the `connectionPools.log` file. The date, time, and other information is recorded. The log file is created in:

<Tomcat-dir>/webapps/qxtendserver/WEB-INF/logs

The log file is configured so that each day at midnight the log file rolls over and archives the existing log file with a date suffix. Then a new log file starts.

QAD QXtend Server Processing Log



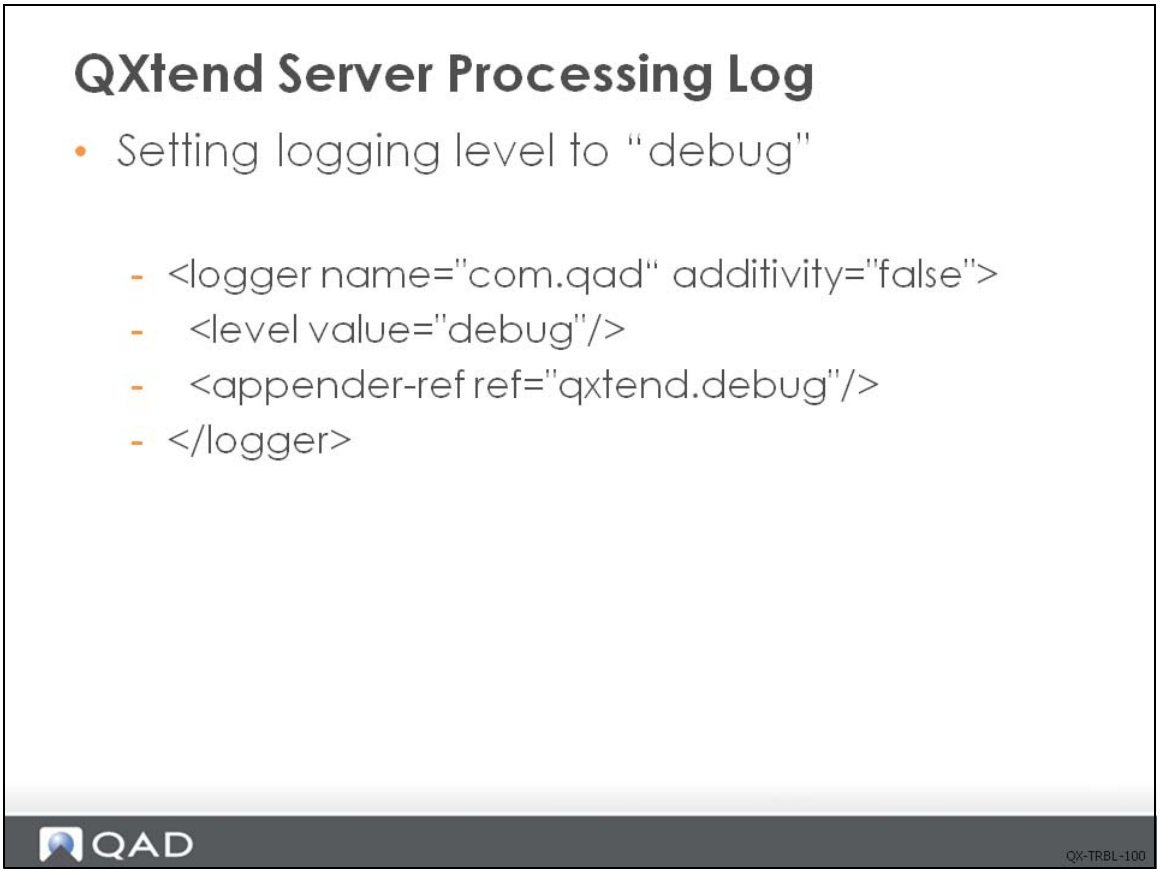
All processing that is internal to the QXtend Server is logged to this file. The logging level of this should always be set to “error” unless otherwise instructed. If the logging setting is set to anything other than error, this log file grows quickly. The debug setting on this file should only be used when problem solving and investigation is taking place.

All activity within the QXtend Server is logged in the `qxtendserver.log` file. The date, time, and other information is recorded. The log file is created in:

```
<Tomcat-dir>/webapps/qxtendserver/WEB-INF/logs
```

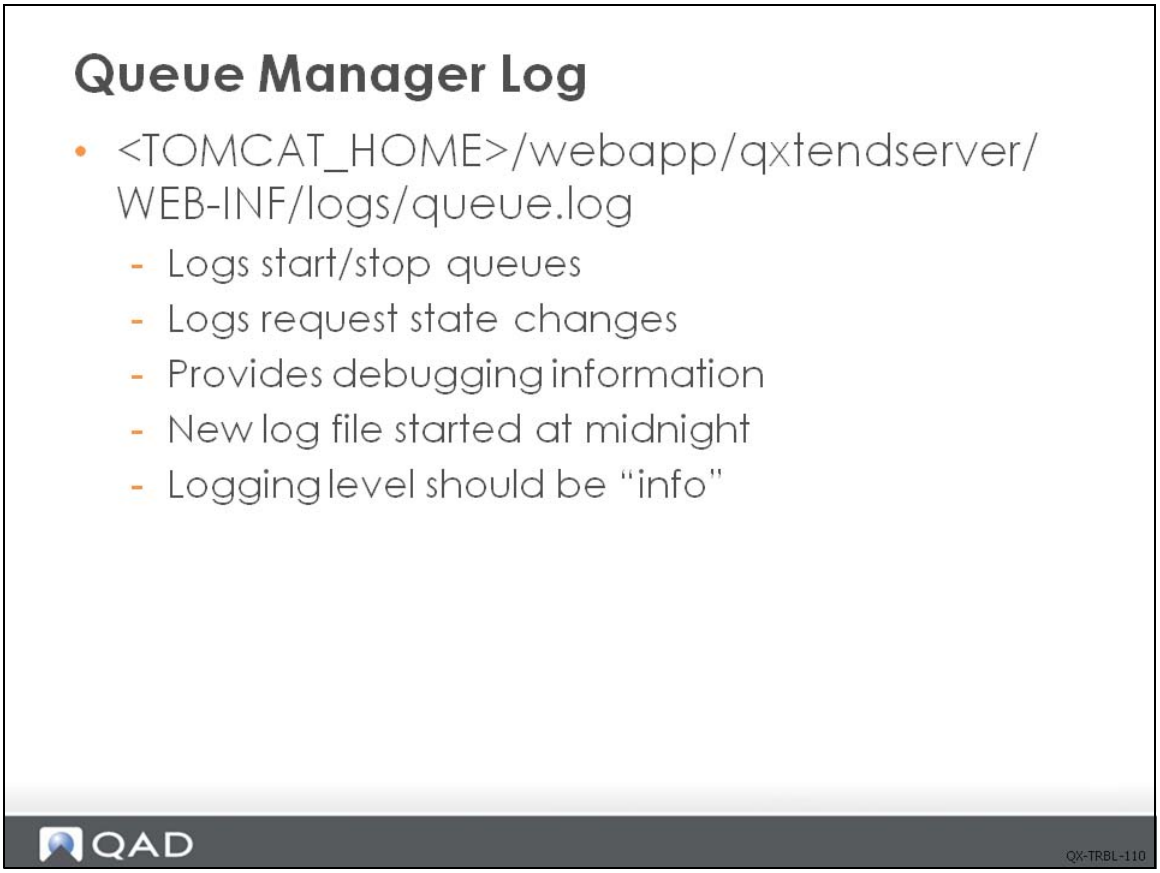
The log file is configured so that each day at midnight the log file rolls over and archives the existing log file with a date suffix. Then a new log file starts.

QAD QXtend Server Processing Log



To enable debug logging for the `qxtendserver.log` file, locate the logger for "`com.qad`" and set the value attribute to "`debug`". This change will then cause QAD QXtend to log debug messages to this log file during processing. This is extremely useful for diagnosing QDoc processing problems.

Queue Manager Log



The Queue Manager log file contains the details of the Queue processing. The log file contains the start/stop of each queue and for each message that is processed every time the request state changes that is also logged. The default setting for the logging level of the queue is "`info`"; debug messages can be received by setting the logging level to "`debug`".

The logger to change the level is:

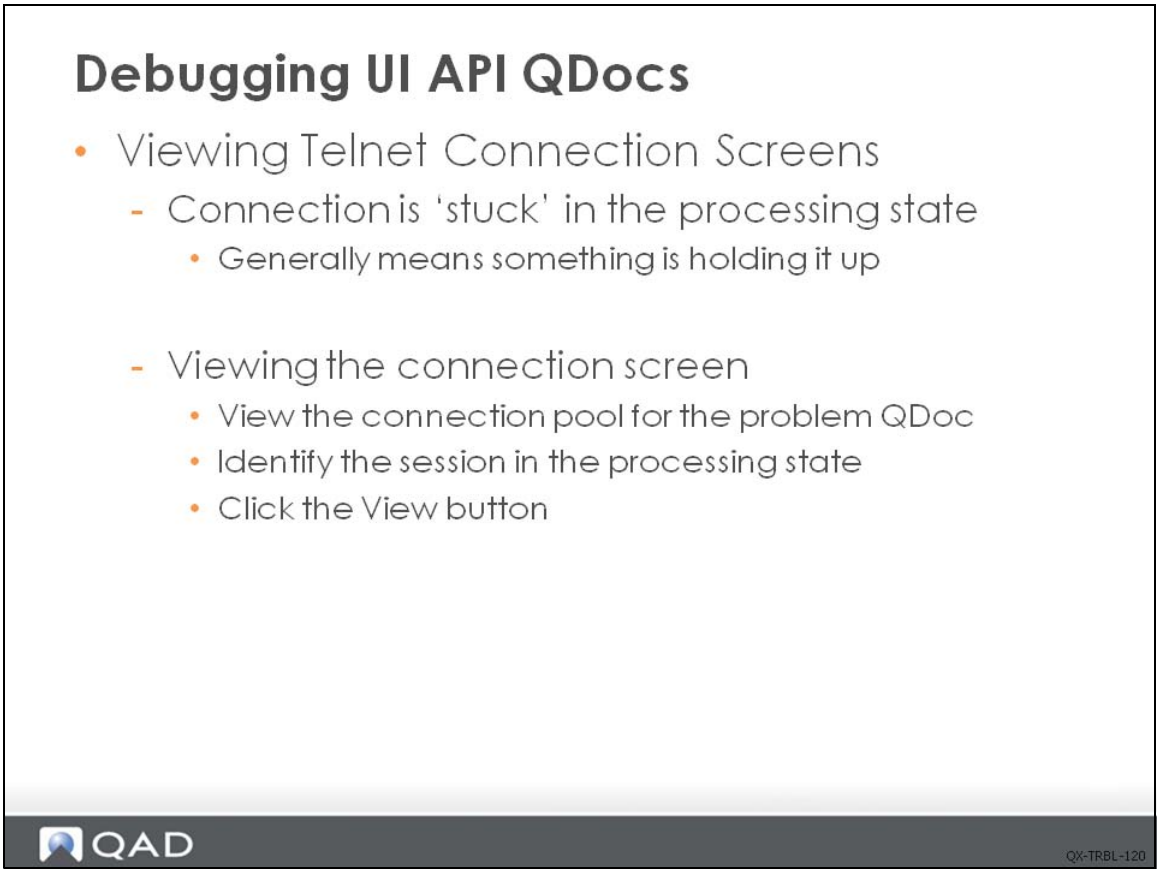
```
<logger name="directoryLoader" additivity="false">
      <level value="info"/>
      <appender-ref ref="asyncqdocDirectoryLoader"/>
</logger>
```

All activity within the QXtend Server is logged in the `queue.log` file; the date and time and other information is recorded. The log file is created in:

```
<Tomcat-dir>/webapps/qxtendserver/WEB-INF/logs
```

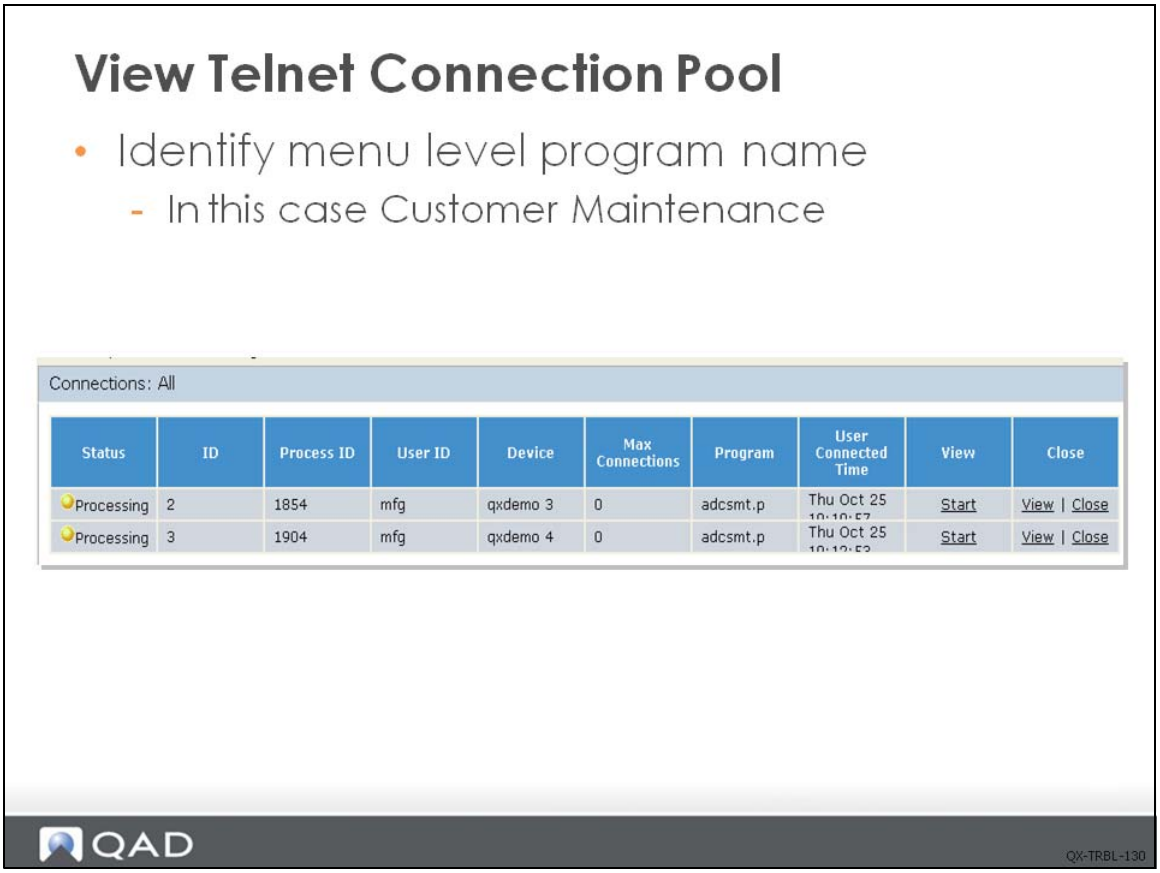
The log file is set so that each day at midnight the log file rolls over and archives the existing log file with a date suffix and starts a brand new log file.

Debugging UI API QDocs



It is possible to view the screen for each telnet connection within the connection pool manager.

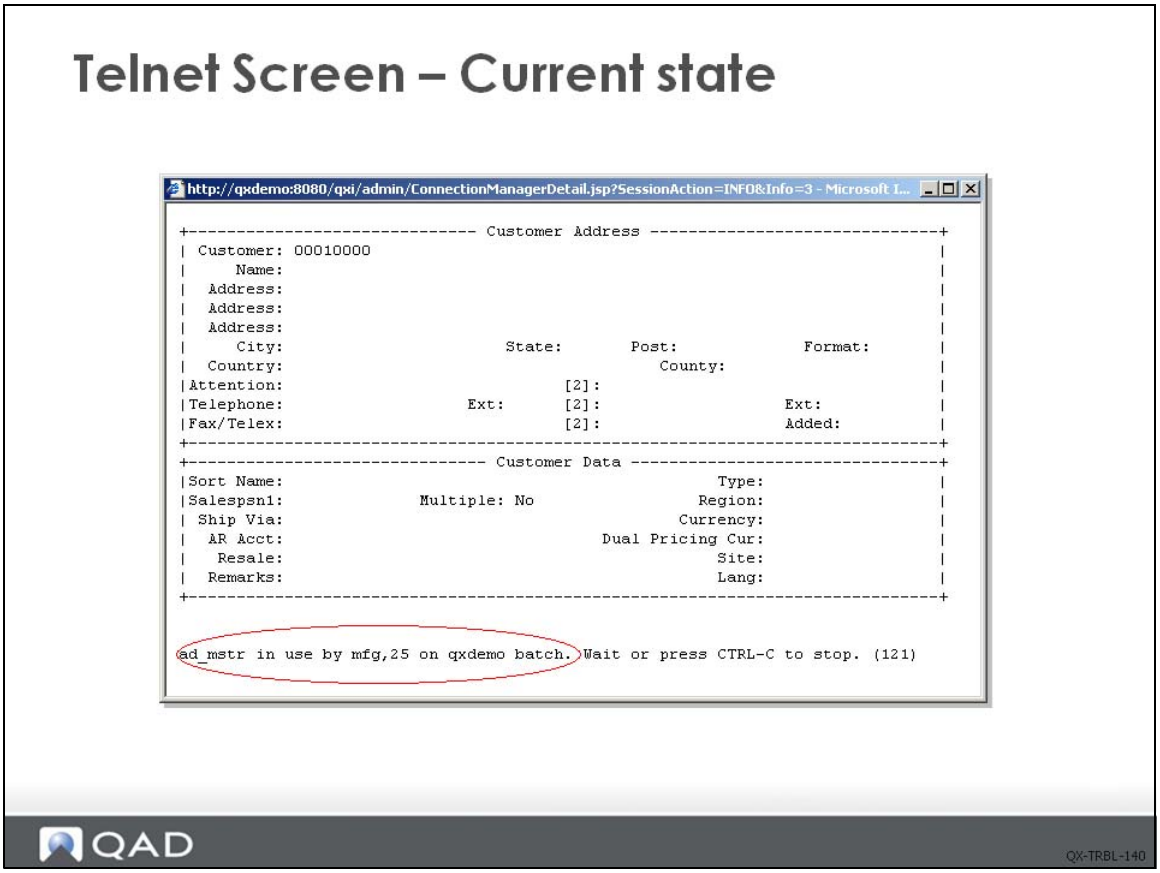
View Telnet Connection Pool



Using the Start link you can debug your QDocs. Choose Start to view the telnet window for the connection being used to import data from a QDoc when that QDoc is being processed. If there is an error in the QDoc, the scrolling of the QDoc on the telnet window halts at the location of the error. When finished testing your connections, click the Stop link, which displays next to the Start link.

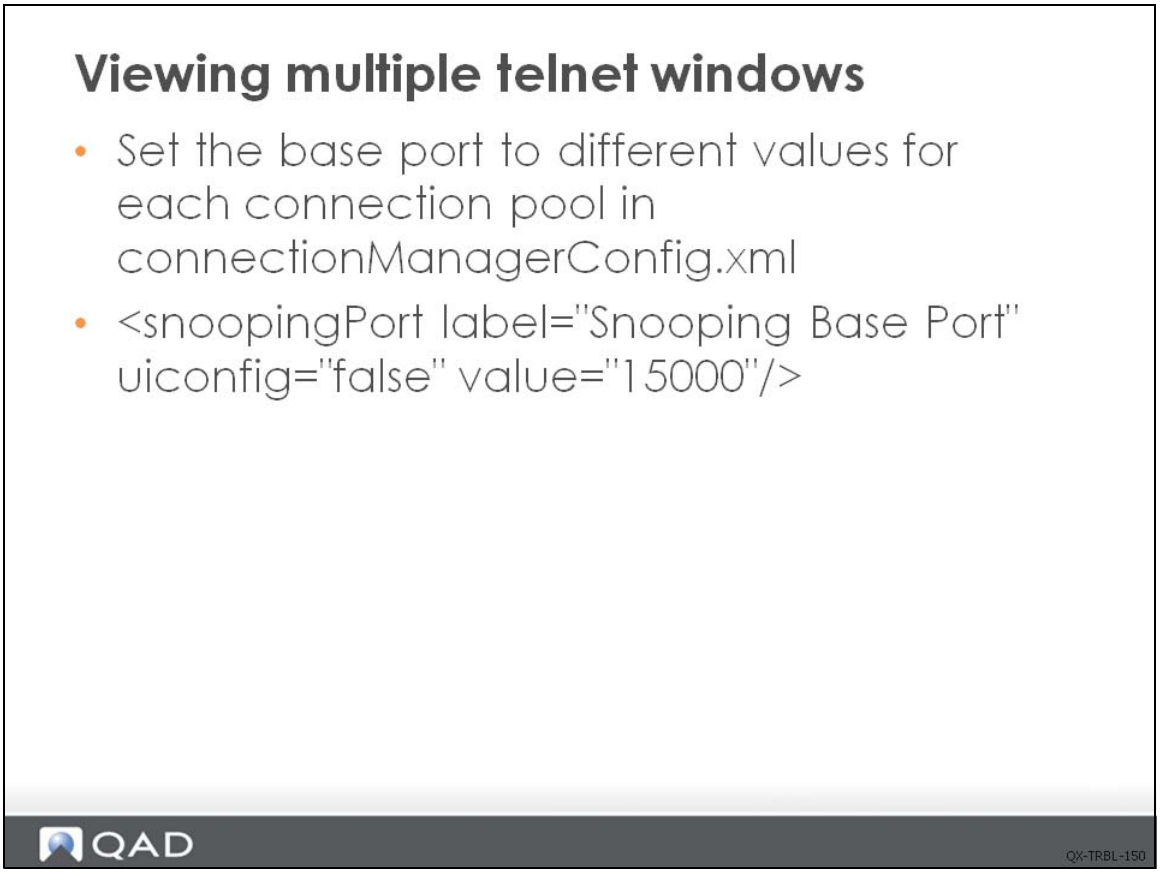
If the QDoc is in the processing state, there is no need to start the telnet viewer—just click the View link on the connection agent in question and view the pop-up screen.

Telnet Screen - Current State



In this example, the processing is being held up because the ad_mstr record is currently locked. Other causes may be unexpected pop-up frames/alert-boxes.

Viewing Multiple Telnet Windows



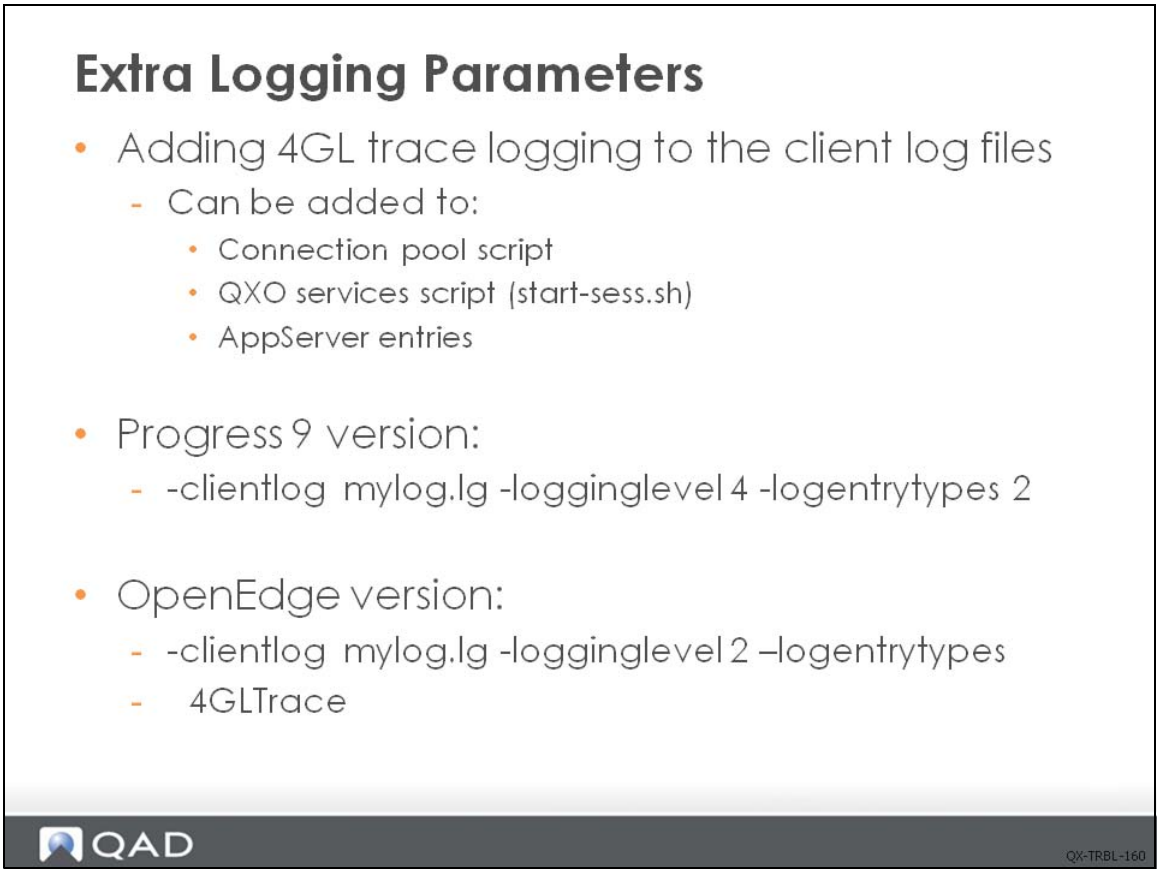
Important Since telnet windows for connections use system resources, you should close any telnet windows not in use.

To view multiple telnet windows—and to avoid network port conflicts while doing so—set the base port to different values for each connection pool in `connectionManagerConfig.xml`.

Note The telnet port that each process agent uses is a combination of the base port and agent ID. For example, for agent ID 2 in this connection pool, the network port used will be 15002.

To be able to use the telnet viewer, you may need to provide access to the specified ports or make sure the ports are accessible in the networking setup of server.

Extra Logging Parameters



Progress client logging can be used to output more detail on the Progress side of the QAD QXtend processing. To do this you add the 4GL Trace parameters to the telnet connection script.

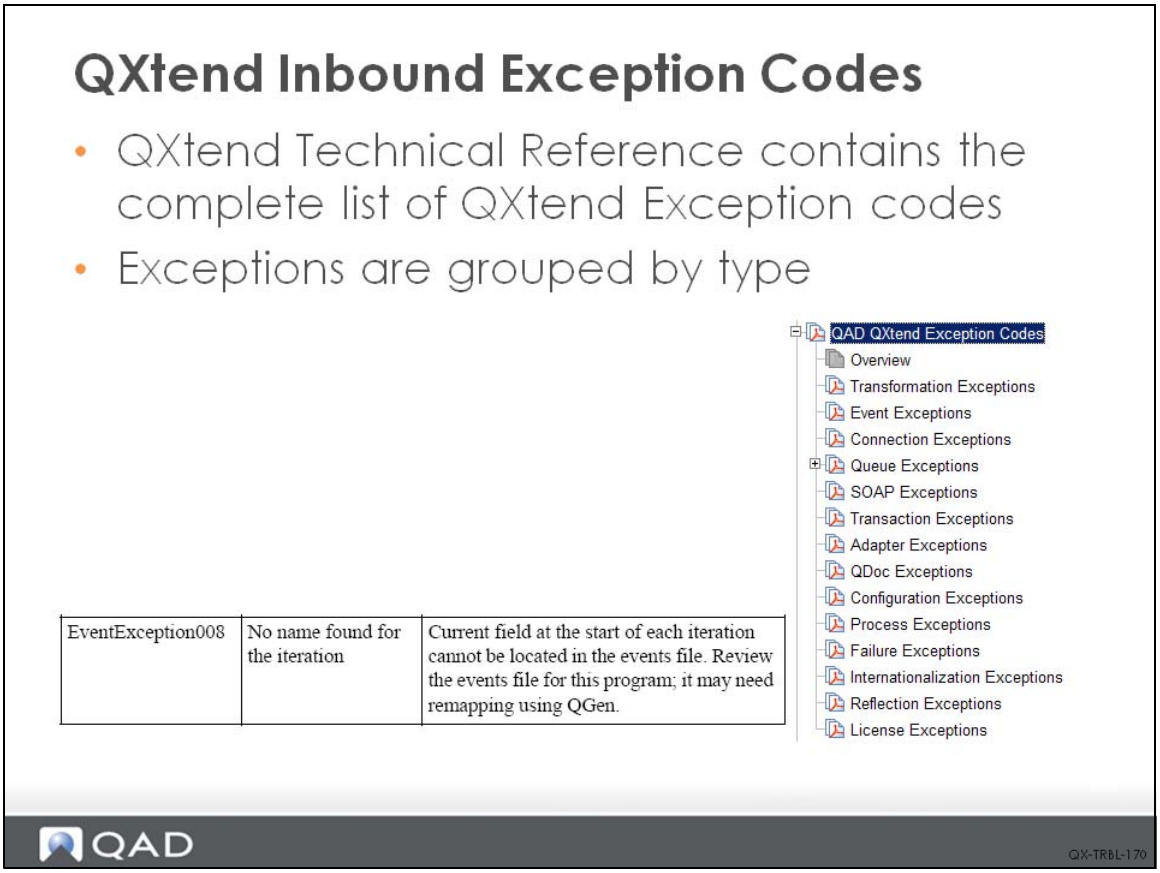
OpenEdge versions

```
$DLC/bin/_progres -clientlog qxilog.txt -logginglevel 2
-logentrytypes 4GLTrace -rereadnolock -c 30 -d mdy -yy 1920 -Bt
350 -D 100 -mmax 3000 -nb 200 -s 128 -noshvarfix -pf
/qad/mfgsvr/Demonstration.pf -p mfwb01aa.p -param mfgwrapper=
true,apimode=true
```

Progress 9 versions

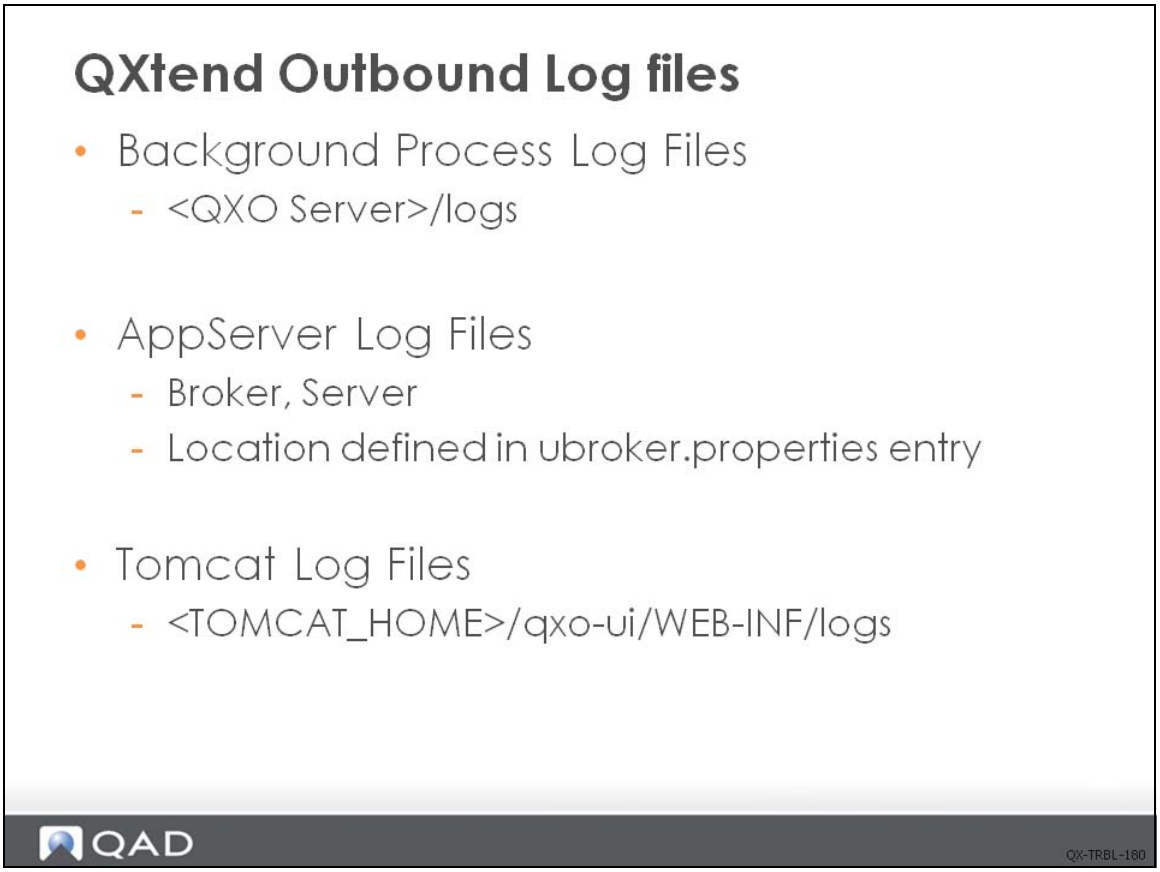
```
$DLC/bin/_progres -clientlog qxilog.txt -logginglevel 4
-logentrytypes 2 -rereadnolock -c 30 -d mdy -yy 1920 -Bt 350 -D
100 -mmax 3000 -nb 200 -s 128 -noshvarfix -pf
/qad/mfgsvr/Demonstration.pf -p mfwb01aa.p -param mfgwrapper=
true,apimode=true
```

QAD QXtend Inbound Exception Codes



The reference section of *Technical Reference: QAD QXtend* lists all exception codes and explains why the exception was raised, as well as suggestions for resolving the problem.

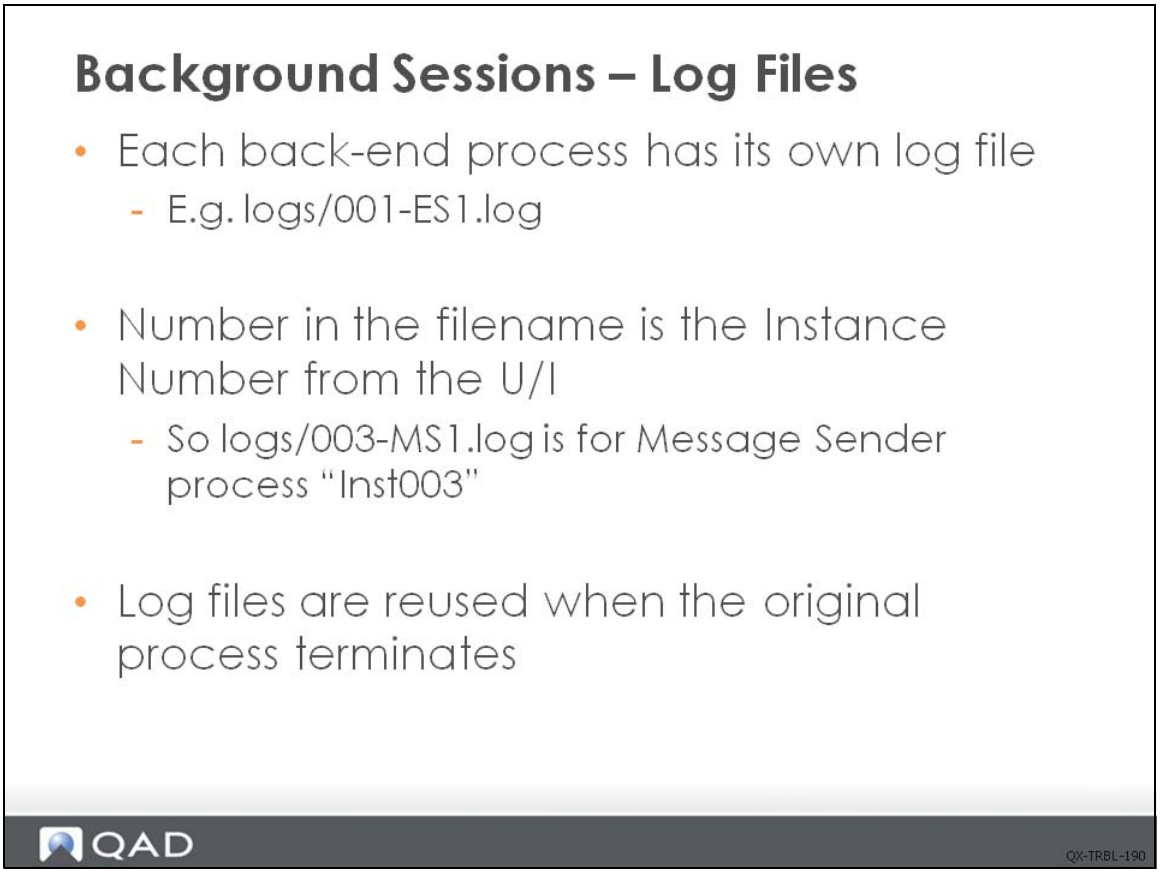
QAD QXtend Outbound Logging



There are three main areas where logging takes place in QXtend Outbound:

- Session log files: the session log files are the log files for the background Progress sessions for the Event Service, Message Publisher, Message Sender and Archive Service
- AppServer log files: the AppServer is used mainly for the communication between the QXO Administration UI (Tomcat/java side) and the QXO DB (OpenEdge side). The AppServer may also be used when implementing calculated fields.
- Tomcat log files: Tomcat is used for housing the QXO Admin UI. As such the logs in `<QXO Webapp>/WEB-INF/logs` only log information related to the UI. No QXO processing information is logged to the Tomcat logs.

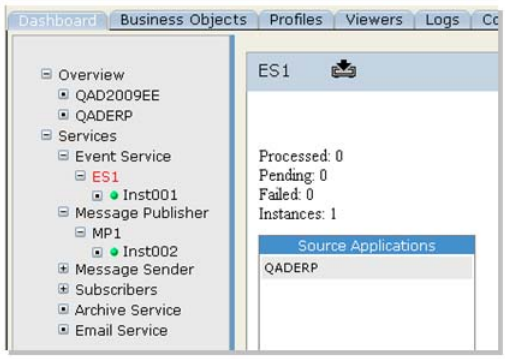
Background Sessions - Log Files



Identify Session Log File

Identify session log file

- Message Publisher is currently Instance 002

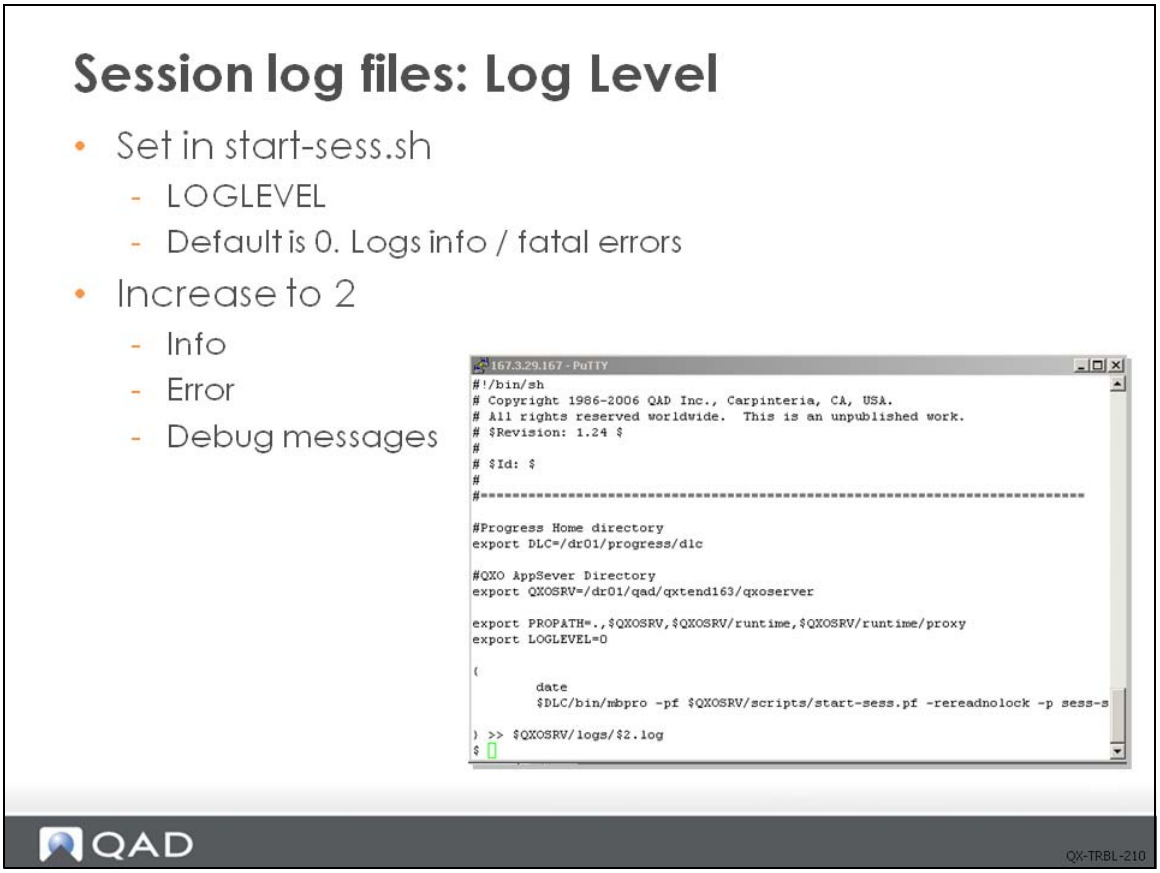


- Corresponds to $QXOSRV/logs/002-MP1.log

QAD

QX-TRBL-200

Session Log Files: Log Level



This `start-sess.sh` can be found under `$QXOSRV/scripts` directory.

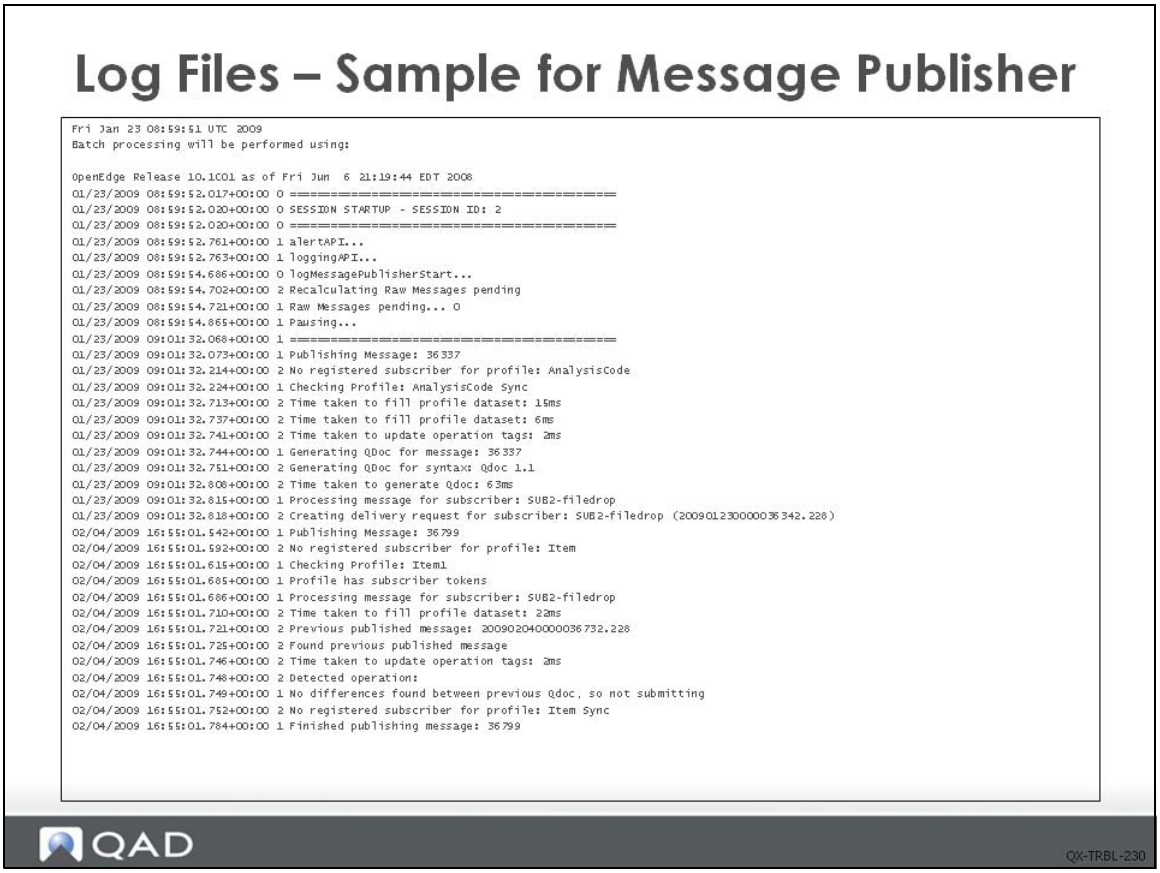
Log Files - Sample for Event Service



This is a sample of the Event Service log file with the logging level increased to 2. The start of the log file shows the session id of the event service. This allows to trace it back to the service instance in the QXO UI. This log file is useful when debugging message extraction problems. When running in debug mode the identifier of the event message is displayed and also the query used on the object including any filter data specified on the Business Object definition.

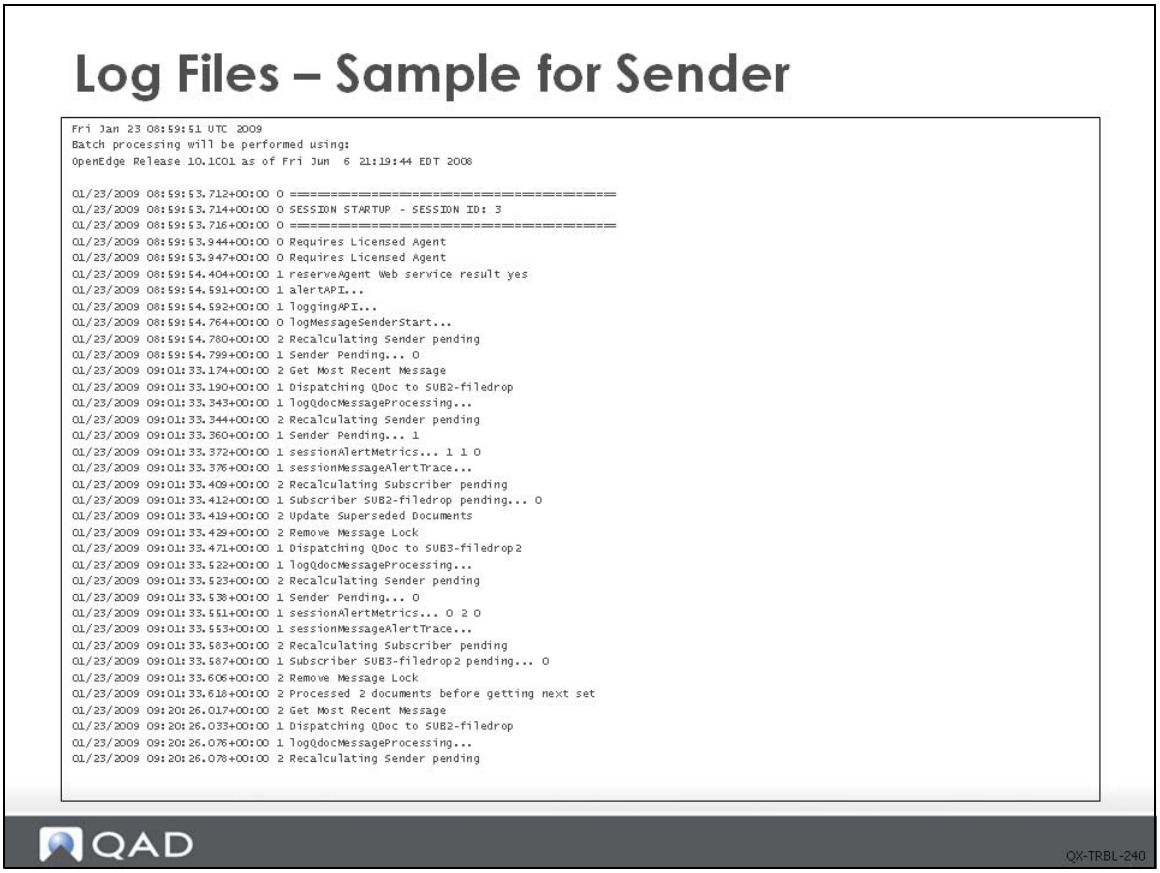
It is worth noting that this log file is only used for extraction type event message and not for events messages that are created using the Direct Data Publish API.

Log Files - Sample for Message Publisher



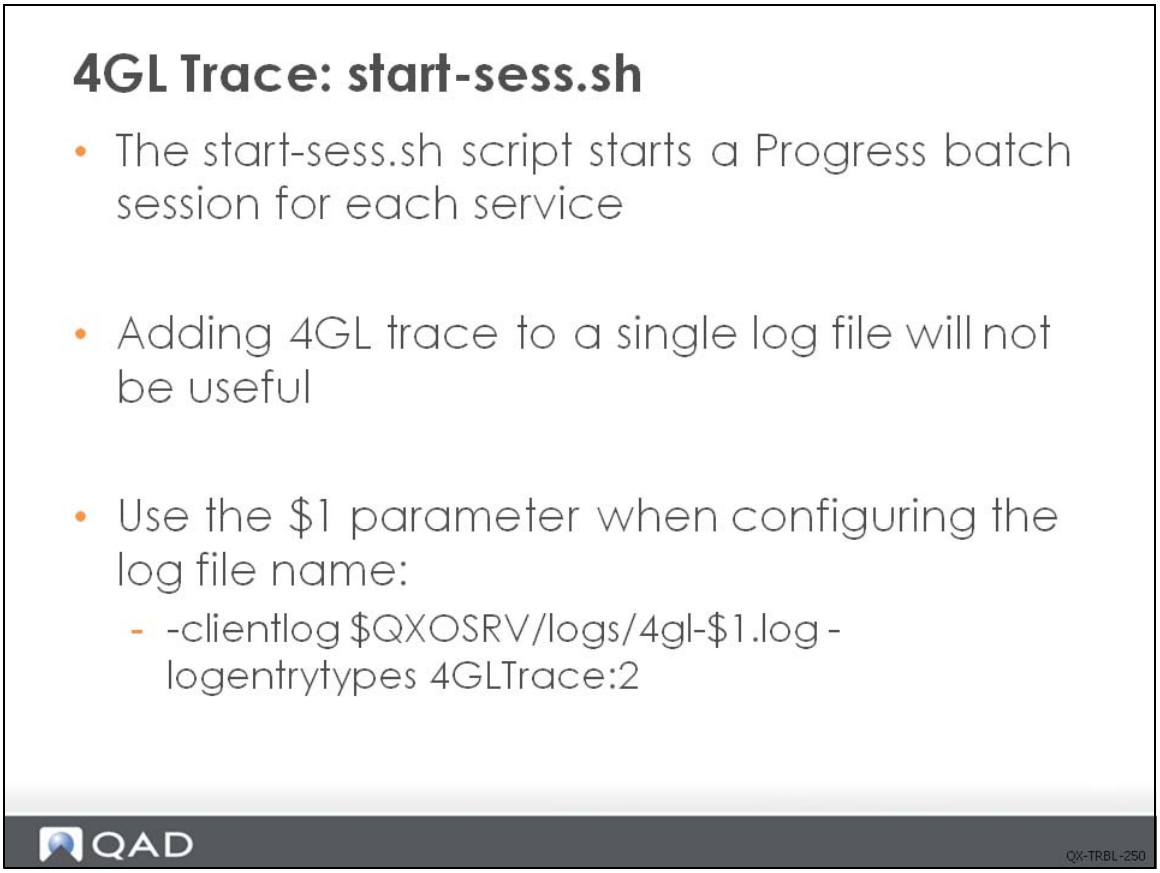
This is a sample of the Message Publisher log file with the logging level increased to 2. The start of the log file shows the session ID of the event service. This allows to trace it back to the service instance in the QXO UI. The Message Publisher is responsible for creating the QDoc profile message, and this log file is useful when debugging the creation of profile messages and delivery requests for each profile message. When running in debug mode, the time taken to create the profile message is logged, along with the name of the profile and any registered subscribers.

Log Files - Sample for Sender



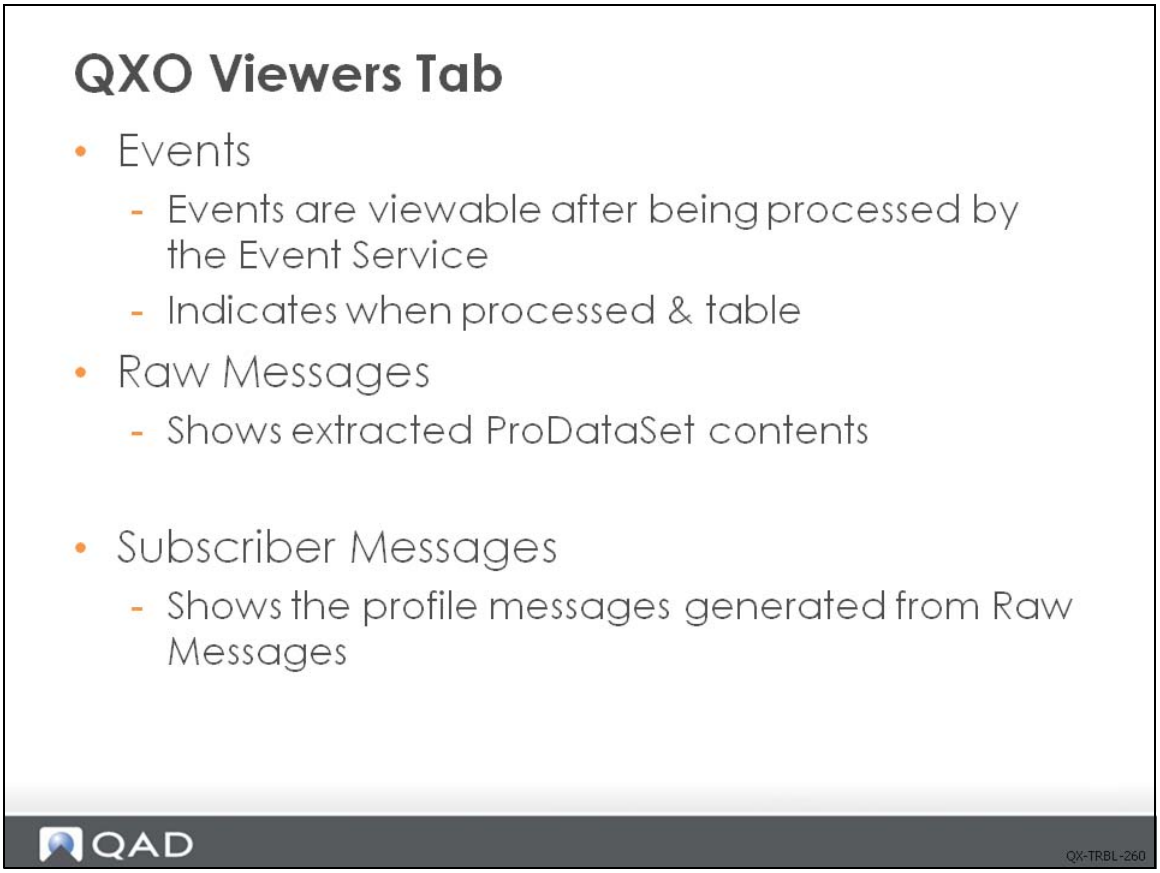
This is a sample of the Message Sender log file with the logging level increased to 2. The start of the log file shows the session id of the event service. This allows to trace it back to the service instance in the QXO UI. The Message Sender is responsible for delivering messages to all registered subscribers. This log file is useful when debugging message delivery problems.

4GL Trace: start-sess.sh



In addition to the LOGLEVEL setting in the `start-sess.sh` script, you can enable the 4GL Trace logging for each of the background services. You will need to use the $1 parameter to separate the trace log files for each of the services. You can add the `-debugalert` option of the logging command to ensure you capture any Progress errors should a session die abruptly.

QXO Viewers Tab

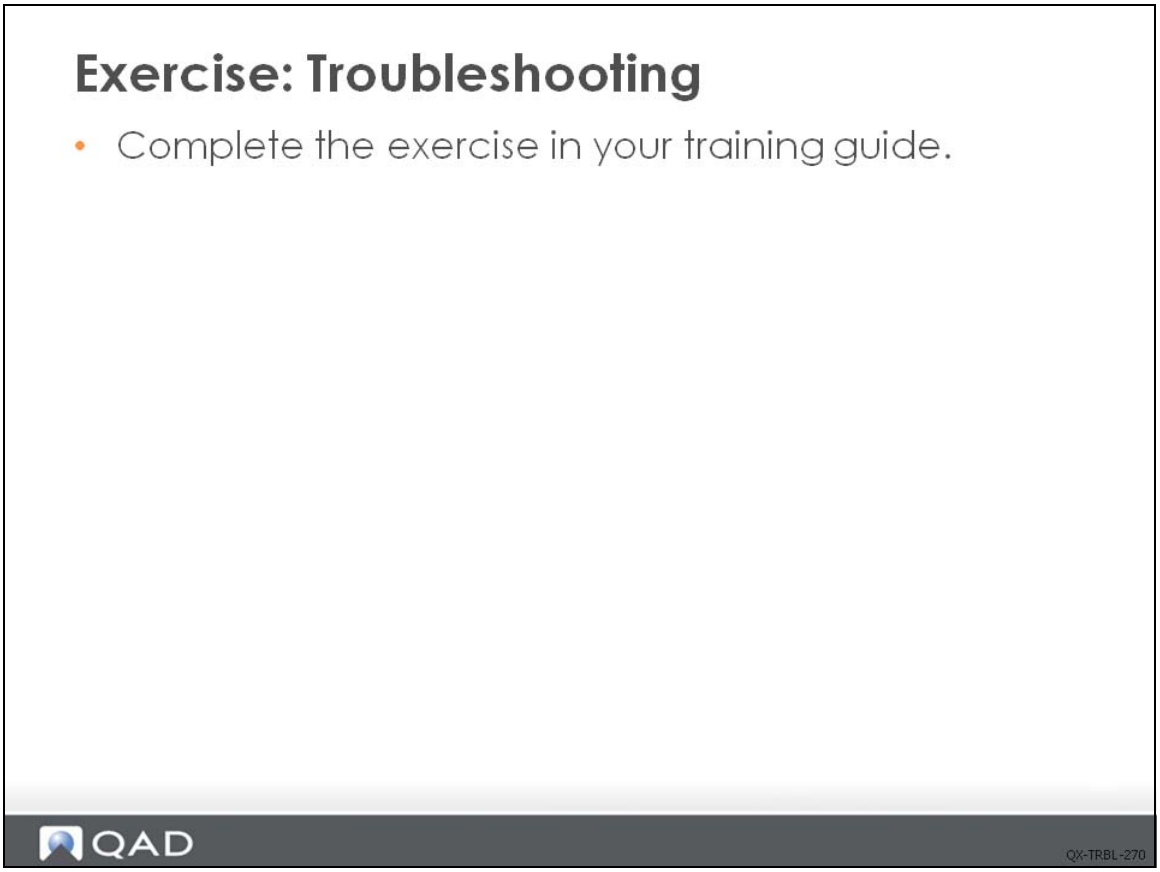


On the Events tab the information available includes the event service instance name, the time stamp of the event record, the source application name, table name and, if applicable, the source domain of the triggered event. You will not see any Direct Data Publish event details as these message use a different API.

The Raw Message tab includes the source application name, source domain, event message identifier (rowid, OID, GUID), and the event type details of the message. Clicking an entry displays the business object event message XML data. This is the data before any associated profile definition is applied to the message.

The Subscriber Messages tab shows entries for the profile name, source application, and the target subscriber. Here you can click an entry and view the request XML (this is the XML message that is sent to the subscriber). You can view the response XML by clicking the View Response button at the bottom of the XML. The response XML contains the response details (if any) returned from the subscribing application.

Exercise: QAD QXtend Diagnostics and Troubleshooting



The following list shows a number of key concepts used in QAD QXtend logging and tools. In each statement below, fill in the correct term from the list.

error	connectionPools.log
session log	Info
qdocResponses.log	QAD QXtend Outbound
qxtendserver.log	Queue Manager
qdocInstall.log	qdocRequests.log
midnight	

1. Each QDoc request that is received by the QAD QXtend Server is logged in the _____ file.
2. Each QDoc response that is returned by the QAD QXtend Server is logged in the _____ file.
3. These log files are set so that each day at _____ the log file rolls over and archives the existing log file with a date suffix and starts a brand new log file.
4. The _____ file logs details about the startup and shutdown of the Environment Manager and all of its managers.
5. All activity with the connection pools is logged in the _____ file; the date, time, and other information is recorded.

6 All processing that is internal to the QAD QXtend Server is logged in the _____ file. To ensure the size of this file does not rapidly become too large, you should set the logging level to _____ unless otherwise instructed.

7 The _____ log file contains the details of the queue processing. The default setting for the logging level of the queue is _____.

8 You can use the 4GL Trace start-sess.sh script to help you troubleshoot log files in _____.

9 The _____ files are the log files for the background Progress sessions for the Event Service, Message Publisher, Message Sender and Archive Service

Lab: Troubleshooting

QAD QXtend 1.7.2 has been installed with both QXI and QXO components. In the previous labs you performed the basic configuration that is required to enable requests to be processed by QXI and QXO. In the following lab exercises you will see how logging is used in QAD QXtend to aid in diagnosing and troubleshooting.

`<LabHomeDirectory> = C:\QXtendTraining\Labs\11-Troubleshooting\`

Logging in QAD QXtend is enabled by default. The logging in QXI is mostly on the Java side. The QXO logging is mostly on the Progress side, remembering that the UI for QXO is housed in Tomcat and does not affect the day-to-day processing in the QXO services.

The exercises in this lab will walk you through enabling the debug log levels and monitoring the log files in QAD QXtend.

1. QAD QXtend Inbound Logging

The logging in QXI uses the log4j framework. The log files generated by QXI are written to the `<tomcat-home>/webapps/<qxi-webapp>/WEB-INF/logs` directory. Logging in QXI occurs by default. However, only information and error messages are logged by default.

To enable debug messages to be output to the log files a configuration file needs to be edited. Follow these steps to enable debug logging in QAD QXtend Inbound:

1. Open PUTTY use the icon on the training Windows machine Desktop.
2. Double-click the qaddemo saved session.
3. Log in as user demo-admin, password qad.
4. Change directory to `/dr01/tomcat/8080/webapps/qxi/WEB-INF/conf`
5. Edit the qxtendlogging.xml file.
   a. vi qxtendlogging.xml.
   b. Change the text below from:

```
<logger name="com.qad" additivity="false">
<level value="error"/>
<appender-ref ref="qxtend.debug"/>
</logger>
```

   to:

```
<logger name="com.qad" additivity="false">
<level value="debug"/>
<appender-ref ref="qxtend.debug"/>
</logger>
```

   c. By setting the level to `debug`, debug messages will start being logged to the qxtendserver.log file.
6. Save your changes to this file.

1.1 Monitoring QAD QXtend Inbound Log Files

The changes that you made in the previous step will take effect after restarting the QXI Web application. The QXI logs directory is located at `<tomcat-home>/webapps/<qxi-webapp>/WEB-INF/logs`.

We need to process some QDoc requests to provide debug messages for analysis. In previous labs you have gained experience processing QDocs. The following steps will take you through processing some requests and then viewing the QAD QXtend Inbound log files:

1. Use one of the requests from the previous labs to process through QXI:

   Customer_Item Update

2. Navigate to the logs directory.
3. Open the WinSCP application that is installed on the Windows image.
4. Open the stored session demo-admin@qaddemo
5. Navigate to the location of the logs directory:

   `/dr01/tomcat/8080/webapps/qxi/WEB-INF/logs`

6. Open the qxtendserver.log file using Wordpad.
7. Search for the string `app=ppcpmt.p`. This is the start of processing the QDoc. By navigating to this part of the log file you can then step through the processing debug messages as the QDoc is processed through the UI API.

Explanation of each step in the debug log is beyond the scope of this course. However it is a useful step in understanding how to locate the start of the QDoc processing and then being able to see any exception error messages. Explanations of the exception error messages are located in the reference section of *Technical Reference: QAD QXtend*.

1.2 QAD QXtend Inbound Connections Viewer

The connection pool for UI API connections contains a built-in connection viewer. Using the Start link you can debug your QDocs. Follow this process to start viewing the telnet connections:

1. Open the QXI page in Internet Explorer: `http://qaddemo:8080/qxi`
2. Navigate to the QADERP UIAPI Connection Pool.
3. View the current connections in the pool.
4. Click the Start link to start the telnet viewer process:


Connections: All

Status	ID	Process ID	User ID	Device	Max Connectio	Program	User Connected Time	View	Close
Idle	1	0	mfg		0	null		Start	Close

5. Click Refresh on the Connections viewer page.
6. You should now see a View link for the connection:



7 Click the View link.

8 A new pop-up window will display. This window will be a direct view on the telnet connection.

9 If there is an error in the QDoc, the scrolling of the QDoc on the telnet window halts at the location of the error. When finished testing your connections, choose the Stop link, which displays next to the Start link.

10 Process the QDocs used in the Queue Manager lab through QXtend and view the connections as they go through the screen.

2. QAD QXtend Outbound Logging

The QAD QXtend Outbound services are on the OpenEdge background sessions. Therefore the logging of the services is handled within the OpenEdge code. This is different to the logging in QAD QXtend Inbound which was Java-based. The QXO Server logs directory is located at:

`/dr01/qadapps/qea/qxtend/qxoserver/logs`

2.1 Identifying QXO Session logs

The session log files for the Event Service, Message Publisher and Message Sender services are written to the QXO Server logs directory and use the naming convention: *<instance number>-<name of service>*`.log`. For example, `001-ES1.log`, `002-MP1.log` and so on.

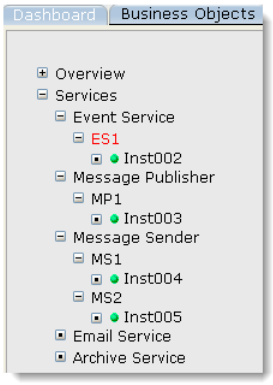
To easily identify which session log corresponds to which service instance, you will need access to the QXO UI to view the sessions that are currently running. Follow this process to identify the service log files in the system:

1 Open the QXI page in internet explorer http://qaddemo:8080/qxo.

2 Select the Dashboard tab.

3 Click the Services link. This displays an overview graph of the services currently running in the system.

4 Expand each of the services links:



Each of the Inst002, Inst003, and so on, has a corresponding session log file in the QXO Server logs directory.

5 Use WinSCP to navigate to the logs directory and identify each of the session log files. The logs directory is located at: `/dr01/qadapps/qea/qxtend/qxoserver/logs`.

6 In this directory there should be files with the names `002-ES1.log`,`003-MP1.log`, `004-MS1.log` and `005-MS2.log`.

7 Open the files and view the level of detail in them.

2.2 Increasing QXO Session Logging Level

The default logging level of the QXO services is 0, which indicates that only Error and Info messages are logged to the session log files. Info messages include details of when services are started and stopped. Error messages occur when processing halts due to a configuration or data error. To increase the logging level, follow this process:

1 Use WinSCP to navigate to the logs directory. Use the demo-admin session that is configured in the WinSCP application.

2 Navigate to the QXO server scripts directory at: `/dr01/qadapps/qea/qxtend/qxoserver/scripts`.

3 Open the start-sess.sh script.

4 Notice the LOGLEVEL variable is currently set to 0.

5 Increase the logging level by setting LOGLEVEL=2.

6 Save the changes to the start-sess.sh file.

7 Note that the change to the logging level setting does not take place automatically. The services need to stopped and started for the setting to take place.

8 Navigate to the Services on the QXO UI and click Stop All.

9 Once the services have stopped, click Start All to start the services again.

10 Using WinSCP, navigate to the logs directory at: `/dr01/qadapps/qea/qxtend/qxoserver/logs`.

11 Open the session log files and view the extra logging details that are present.

12 Using the Generalized Codes configuration setup in a previous lab (06 Business Objects and Profiles), process some Generalized Codes records. You can do this by logging into QAD EE and creating some Generalized Codes records.

13 View the session log files again using WinSCP navigate to the logs directory at:

```
/dr01/qadapps/qea/qxtend/qxoserver/logs
```

14 Open the session log files and view the extra logging details that are present.