User Guide
QAD QXtend

QAD QXtend Outbound
QAD QXtend Inbound
QAD QXtend Licensing
Reference

70-3093A
QAD QXtend 1.7.1
MFG/PRO eB, eB2, eB2.1, QAD Enterprise Applications Standard and Enterprise Editions
March 2011

Qxtend_UG_v010701.pdf/hes/hes

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

Contents

Section 1

QAD QXtend Outbound

This section contains information on implementing and using QAD QXtend Outbound (QXO).

Chapter 1

QAD QXtend Outbound Overview

This chapter provides a functional and architectural overview of QAD QXtend Outbound (QXO) and also provides an overview of QXO implementation.

Application Overview

QXO is a component of the QAD QXtend interoperability framework. This framework provides a standardized data interface between QAD products, and between QAD products and external systems. The framework consists of:

- QXO, which exports user-configurable business objects as XML QDocs out of QAD products such as QAD Enterprise Applications to external subscribers. QXO is described in this section of this reference guide.
- QAD QXtend Inbound (QXI), which imports SOAP-compliant XML QDocs from external applications into QAD Enterprise Applications and other QAD applications. For details see “QAD QXtend Inbound” on page 143.

This interoperability framework replaces QAD products and solutions including CIM, Q/LinQ, and Data Synchronization. See “Glossary” on page 415 for definitions of terms and abbreviations.

QXO exports business objects from QAD Enterprise Applications versions eB and above. This user guide describes the implementation of QXO for QAD Enterprise Applications. Details about using and implementing QXO with other QAD applications can be found in the guides for those applications.

Because QXI and QXO have different data requirements, inbound and outbound QDocs are not always equivalent. Inbound QDocs provide input to QAD Enterprise Applications in the same way a user would. The data is input field-by-field through the user interface, but the input is automated. Therefore, inbound QDocs require a rigid data sequence and are limited to a single entry program, such as Sales Order Maintenance.

Outbound QDocs usually include data from multiple QAD Enterprise Applications tables, and possibly from multiple screens, in order to provide external subscribers with a comprehensive business object.

Note Data originating from the Financials module in QAD EE—due to the module’s underlying system architecture—is provided to QXO as a raw business object. As such there is no need to extract data for these business objects from the QAD Financials database.

Business Objects and Outbound QDocs

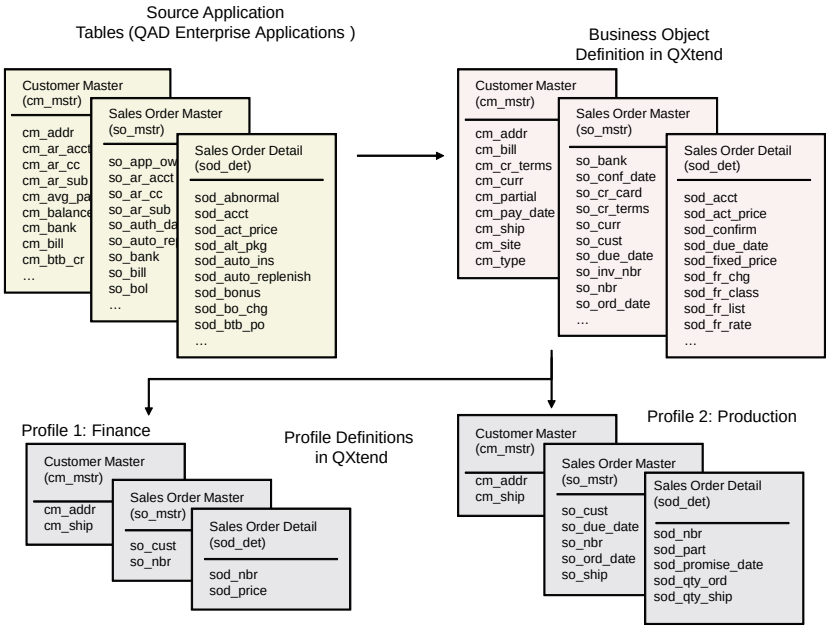
All outbound QDocs are defined by an underlying business object. A business object usually consists of data from a related set of tables required from a source QAD application. One such set is the sales order business object, which consists of sales order header and line data; ship-to, sold-to, and bill-to customer information; Intrastat data; tax details; and comments.

Each business object is defined in the QXO administrative interface, which lets you select the tables and fields your target applications—your subscribers—need. However, all subscribers do not require the same elements from any given business object. To tailor business object output for multiple subscribers, you define profiles that filter and qualify business object data for one or more subscribers.

Business objects that are associated with a direct data publish (DDP) source application type are passed directly to QXO. For details see “Direct Data Publish” on page 13.

Figure 1.1 shows the basic data flow of data from QAD Enterprise Applications tables to the profiles that provide the template for outbound QDocs for specific subscribers.

Fig. 1.1
Data Flow from QAD Enterprise Applications to Profiles



With the current release of QXO, the following 36 business objects are shipped with the product.

Table 1.1
Standard QXO Business Objects for QAD Enterprise Applications

Alternate Unit of Measure	Inventory Status	Sales Order Price List
Analysis Code	Inventory Transaction	Scheduled Sales Order
Analysis Code Link	Invoice History	Service Category
Analysis Code Selection	Item	Service Support Call
Carrier	Item Site Cost	Service Support End User
Cost Set	Master Comment	Service Support Engineer
Customer	Product Line	Service Support Work Code
Customer Item	Product Structure	Shipper
Customer Schedule	Product Structure Code	Ship-To Customer
Department	Purchasing Price List	Site
Forecast	Routing	Supplier Item
Generalized Codes	Sales Order	Work Center

These business objects reference data in the tables listed in Table 1.3. Tables shown in Table 1.2 use database triggers or business events; if there is no business event indicated, a database trigger is used.

Table 1.2
Tables with Programmatic Triggers

Business Event	Table Name	Description	QAD EE	QAD SE
	abs_mstr	ASN/BOL/Shipper Master	✓	✓
	absd_det	Shipment Line Item Detail	✓	✓
	absl_det	Shipment Detail Line Charges	✓	✓
	absr_det	Shipment Requirement Detail	✓	✓
AccountMaintenance	ac_mstr	Account Master		✓
CarrierAddressMaintenance			✓	
CurrencyAccountMaintenance	acdf_mstr	Account Default Master		✓
	actid_det	Activity ID Detail		
	ad_mstr	Address Master	✓	✓
	an_mstr	Analysis Code Master	✓	
	anl_det	Analysis Link Code Detail	✓	
	ans_det	Analysis Code Selection Detail	✓	
	asc_mstr	Account/Sub-Account/Cost Center Master		
	aterr_mstr	Audit Trail Error Master		
	attmp_mstr	Audit Trail Temporary Master		
	bom_mstr	Product Structure (Bill of Material) Master	✓	✓
	ca_mstr	Service/Support Call Master	✓	✓
	cc_mstr	Cost Center Master		
	ccd_mstr	Service/Support Call Fault Code Master	✓	✓
	cd_det	Master Comments	✓	✓
CustomerMaintenance	cm_mstr	Customer Master	✓	✓
ShipToMaintenance	cm_mstr	Customer Master	✓	✓
	cmt_det	Transaction Comments	✓	✓
	code_mstr	Generalized Code Master	✓	✓
	com_mstr	Commodity Code Master		✓
	cp_mstr	Customer Item Master	✓	✓
	cs_mstr	Cost Set Master	✓	
CreditTermsMaintenance	ct_mstr	Credit Terms Master		✓

Business Event	Table Name	Description	QAD EE	QAD SE
CommodityCodeMaster	com_mstr	Commodity Code Master	✓	
CommodityCodeMaintenance				✓
	ctd_det	Credit Terms Detail		
CountryMaintenance	ctry_mstr	Country Master	✓	✓
CurrencyAccountMaintenance	cu_mstr	Currency Master		✓
CurrencyMaintenance				✓
	dom_mstr	Domain Master		
	dpt_mstr	Department Master	✓	
	dy_mstr	General Ledger Daybook Master		
	dybs_mstr	Daybook Set Master		
	egt_mstr	Service/Support Engineer Tracking Master	✓	✓
	emp_mstr	Employee Master		
	en_mstr	Entity Master		
	eng_mstr	Service/Support Engineer Master	✓	✓
	eu_mstr	Service/Support End User Master	✓	
	eud_det	Service/Support End User Detail		
ExchangeRateMaintenance	exr_rate	Exchange Rate Master		✓
FormatPositionMaintenance				✓
GeneralizedCodeMaintenance			✓	✓
	fcs_sum	Forecast Summary	✓	✓
	fsc_mstr	Service Category Master	✓	✓
	fwk_mstr	Service/Support Work Code Master	✓	✓
	gl_ctrl	System/Account Control		
CalendarMaintenance	glc_cal	General Ledger Calendar		✓
	glcd_det	General Ledger Calendar Detail		
	idh_hist	Invoice History Detail	✓	✓
	ied_det	Import/Export Detail	✓	✓
	ie_mstr	Import/Export Master	✓	✓
	ih_hist	Invoice History Master	✓	✓
InventoryStatusMaintenance			✓	✓
InvoicePost			✓	✓
ItemDataMaintenance			✓	✓

Business Event	Table Name	Description	QAD EE	QAD SE
ItemInventoryMaintenance			✓	✓
ItemMaintenance	in_mstr	Inventory Master	✓	✓
ItemSiteInventoryMaintenance			✓	✓
	isb_mstr	Service/Support Installed Base Item Master	✓	✓
	isd_det	Inventory Status Detail	✓	
	is_mstr	Inventory Status Master	✓	✓
	itm_det	Service/Support Call Item Detail	✓	✓
	lacd_det	Logistics Accounting Charge Detail	✓	✓
	lng_mstr	Language Master		
	ls_mstr	Address List Detail	✓	
	nr_mstr	Number Range Master		
	pc_mstr	Price List Manager	✓	
	pid_det	Price List Detail	✓	✓
	pi_mstr	Price List Master	✓	✓
	pl_mstr	Product Line Master	✓	✓
	pod_det	Purchase Order Detail		✓
	po_mstr	Purchase Order Master		✓
ProductLineMaintenance			✓	✓
	ps_mstr	Product Structure Master	✓	✓
	pt_mstr	Item Master	✓	✓
	ptp_det	Item Planning Detail	✓	✓
	qaddb_ctrl	Database Control for QADDB		
	rnd_mstr	Rounding Method Master		
	ro_det	Routing Operation Detail	✓	✓
	schd_det	Release Management Schedule Detail	✓	✓
	sch_mstr	Release Management Schedule Master	✓	✓
	sct_det	Cost Simulation Total Detail	✓	✓
	scx_ref	Scheduled Order Cross-Reference	✓	✓
	si_mstr	Site Master	✓	✓
SalesOrderMaintenance	so_mstr	Sales Order Master	✓	✓
SalesOrderPrint	so_mstr	Sales Order Master		✓
SalesOrderShip	so_mstr	Sales Order Master		✓

Business Event	Table Name	Description	QAD EE	QAD SE
	sob_det	Sales Order Configuration Bill Detail	✓	✓
	sod_det	Sales Order Detail	✓	✓
	sodlc_det	Sales Order Detail Line Charges	✓	✓
	spt_det	Cost Simulation Item Detail	✓	✓
	tr_hist	Inventory Transaction History	✓	✓
	tx2d_det	Tax Detail	✓	✓
	um_mstr	Unit of Measure Conversion Master	✓	
	udd_det	User Domain Detail		
	usrgd_det	User Group Detail		
ShipToMaintenance				✓
ShipperWorkbench			✓	✓
ScheduledOrderMaintenance			✓	✓
SupplierMaintenance	vd_mstr	Vendor Master	✓	✓
	vp_mstr	Supplier Item Master	✓	
	wc_mstr	Work Center Master	✓	
	wo_mstr	Work Order Master	✓	✓
	wod_det	Work Order Detail	✓	✓
	wr_route	Word Order Routing	✓	✓

Each of the tables in Table 1.3 is defined in QXO as an event type. For these tables, schema triggers are loaded during installation and can be activated as needed. The tables vary slightly by QAD Enterprise Applications release. A check mark indicates that, by default, a trigger is provided in the database schema.

Table 1.3
Tables with Schema Triggers

Table Name	Description	eB	eB2	eB2.1	QAD SE	QAD EE
absc_det	Shipment Carrier Detail					✓
absd_det	Shipment Line Item Detail		✓	✓	✓	✓
absl_det	Shipment Detail Line Charges		✓	✓	✓	✓
absr_det	Shipment Requirement Detail	✓	✓	✓	✓	✓
abs_mstr	ASN/BOL/Shipper Master	✓	✓	✓	✓	✓
acdf_mstr	Account Default Master	✓	✓	✓	✓	
ac_mstr	Account Master	✓	✓	✓	✓	
ad_mstr	Address Master	✓	✓	✓	✓	✓
anl_det	Analysis Code Link Detail					✓
ans_det	Analysis Code Selection Detail					✓

Table Name	Description	eB	eB2	eB2.1	QAD SE	QAD EE
an_mstr	Analysis Code Master					✓
bom_mstr	Product Structure (Bill of Material) Master	✓	✓	✓	✓	✓
ca_mstr	Service/Support Call Master	✓	✓	✓	✓	✓
ccd_mstr	Service/Support Call Fault Code Master	✓	✓	✓	✓	✓
cd_det	Master Comments	✓	✓	✓	✓	✓
cm_mstr	Customer Master	✓	✓	✓	✓	✓
cmt_det	Transaction Comments	✓	✓	✓	✓	
code_mstr	Generalized Codes Master	✓	✓	✓	✓	✓
com_mstr	Commodity Code Master	✓	✓	✓	✓	
cp_mstr	Customer Item Master	✓	✓	✓	✓	✓
cs_mstr	Cost Set Master					✓
cu_mstr	Currency Master	✓	✓	✓	✓	
dpt_mstr	Department Master					✓
egt_mstr	Service/Support Engineer Tracking Master	✓	✓	✓	✓	✓
eng_mstr	Service/Support Engineer Master	✓	✓	✓	✓	✓
eu_mstr	Service/Support End User Master	✓	✓	✓	✓	✓
fcs_sum	Forecast	✓	✓	✓	✓	✓
fsc_mstr	Service Category Master	✓	✓	✓	✓	✓
fwk_mstr	Service/Support Work Code Master	✓	✓	✓	✓	✓
glc_cal	General Ledger Calendar	✓	✓	✓	✓	
idh_hist	Invoice History Detail	✓	✓	✓	✓	✓
ie_mstr	Import/Export Master	✓	✓	✓	✓	✓
ied_det	Import/Export Detail	✓	✓	✓	✓	✓
ih_hist	Invoice History Master	✓	✓	✓	✓	✓
in_mstr	Inventory Master	✓	✓	✓	✓	✓
is_mstr	Inventory Status Master	✓	✓	✓	✓	✓
isb_mstr	Service/Support Installed Base Item Master	✓	✓	✓	✓	✓
isd_det	Inventory Status Detail					✓
itm_det	Service/Support Call Item Detail	✓	✓	✓	✓	✓
lacd_det	Logistics Accounting Charge Detail		✓	✓	✓	✓
ls_mstr	Address List Detail	✓	✓	✓	✓	✓
pc_mstr	Price List Master - Purchasing					✓
pi_mstr	Price List Master	✓	✓	✓	✓	✓
pid_det	Price List Detail	✓	✓	✓	✓	✓

Table Name	Description	eB	eB2	eB2.1	QAD SE	QAD EE
pl_mstr	Product Line Master	✓	✓	✓	✓	✓
pod_det	Purchase Order Detail	✓	✓	✓	✓	
po_mstr	Purchase Order Master	✓	✓	✓	✓	
ps_mstr	Product Structure Master	✓	✓	✓	✓	✓
pt_mstr	Item Master	✓	✓	✓	✓	✓
ptp_det	Item Planning Detail	✓	✓	✓	✓	✓
ro_det	Routing Operation Detail	✓	✓	✓	✓	✓
sch_mstr	Release Management Schedule Master	✓	✓	✓	✓	✓
schd_det	Release Management Schedule Detail	✓	✓	✓	✓	✓
sct_det	Cost Simulation Total Detail	✓	✓	✓	✓	✓
scx_ref	Scheduled Order Cross-Reference	✓	✓	✓	✓	✓
si_mstr	Site Master	✓	✓	✓	✓	✓
sob_det	Sales Order Configuration Bill Detail	✓	✓	✓	✓	✓
sod_det	Sales Order Detail	✓	✓	✓	✓	✓
sodlc_det	Sales Order Detail Line Charges		✓	✓	✓	✓
so_mstr	Sales Order Master	✓	✓	✓	✓	✓
spt_det	Cost Simulation Item Detail	✓	✓	✓	✓	✓
tr_hist	Inventory Transaction History	✓	✓	✓	✓	✓
tx2d_det	Tax Detail	✓	✓	✓	✓	✓
um_mstr	Alternate Unit of Measure Master					✓
vd_mstr	Supplier Master	✓	✓	✓	✓	✓
vp_mstr	Vendor Item Master					✓
wc_mstr	Work Center					✓
wo_mstr	Work Order Master	✓	✓	✓	✓	✓
wod_det	Work Order Detail	✓	✓	✓	✓	✓
wr_route	Work Order Routing	✓	✓	✓	✓	✓

Sequencing Business Dependencies

Because QXO supports multiple event services in a distributed environment, it is possible to load data messages in an incorrect sequence. For example, a sales order may be sent through before the new or revised customer record for that order. In certain scenarios, it is possible for you to enforce correct chronological sequencing of data messages to avoid this kind of situation.

By assigning business objects to a business object group, you can ensure that the business objects and their tables will be processed and published in the same order they are set up in the group, provided that all the active event types in the group are registered with the same event service. For

example, place the Customer and Sales Order business objects in a business object group and register all related active event types with the same event service so that if an event is generated for both these objects, the customer data will be always extracted prior to the sales order data.

However, if active event types in a business object group are not registered with the same event service, business objects in the group will be processed by different event services, which defeats the purpose of business object groups since you cannot guarantee that data of designated business objects will be processed as set up in the group in a single-thread manner. This gives rise to a warning message in configuration validation.

See “Adding Source Application Business Object Groups” on page 27.

For information on event type registration, see “Registering Event Types with an Event Service” on page 35.

Business object groups are set up as part of the source application configuration. Each business object can belong to only one group.

Note Business object groups are not defined for business objects that employ DDP.

Business Events

Previous versions of QXtend Outbound only used database triggers as events; in version 1.6 a business process can also be the event that triggers a business object to be extracted. The term for event-based notification triggers is *business events*.

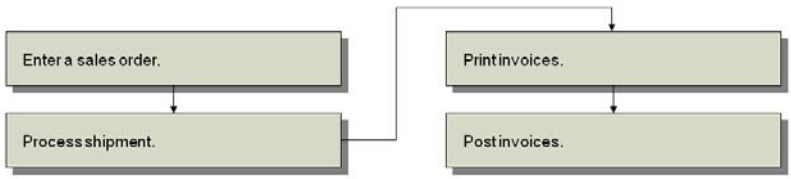
Business events enhance your control over when a business object is extracted, including the ability to trigger processing at crucial stages within a workflow.

The term *workflow* typically denotes a built-in automated approval process that can be applied to any record (or document) created in the system. Essentially a workflow depicts a sequence of operations within a business process. Workflow can be used for various purposes including:

- Approve supplier invoices
- Release invoices for payment
- Notify stakeholders of changes to accounts, GL data, and customer or supplier data
- Escalate overdue customer invoices

For example, a sales order typically has a well-defined lifecycle, from entering the sales order into the system, through processing the shipment, then printing and posting related invoices, as shown in Figure 1.2.

Fig. 1.2
Sales Order Lifecycle



Using business events you could extract a business object for processing if a quantity value or ship-to location is changed on a line item, or promote the business object into a different workflow stage.

In non-direct data publish source applications, events can be database triggers (raised against a specific database table) or business events (raised against business objects). In DDP source application types, business events are the only event type.

You can configure profiles to receive only the business object messages you want based on the business events raised by a source application. For details, see “Modifying Profiles” on page 89. You also can configure subscribers to receive specific profile messages based on events raised by the source application. For details, see “Configuring Subscriber Profiles” on page 45.

Direct Data Publish

QAD EE uses the QAD Reference Architecture technology, which supports true business object functionality where the change in state of a business object can be detected—it is not necessary to poll the `qxevents` database for changes and extract the data. Instead the raw business object event data is saved in an XML file and passed directly to QXO via an event message API—the resulting QDoc is published without the need for further processing.

This ability is known as *direct data publish* (DDP), and enables data to be synchronized between one instance of the Financials module and another, or between the Financials module and an external application.

Since DDP source application types post event data directly to QXO, no event service is required. The QDoc is published using the standard message publisher functionality.

DDP employs a direct data publishing API to process event messages and schemas originating from a source application type that employs DDP. The direct data publishing API utilizes the service interface adapter API to input data into the target application.

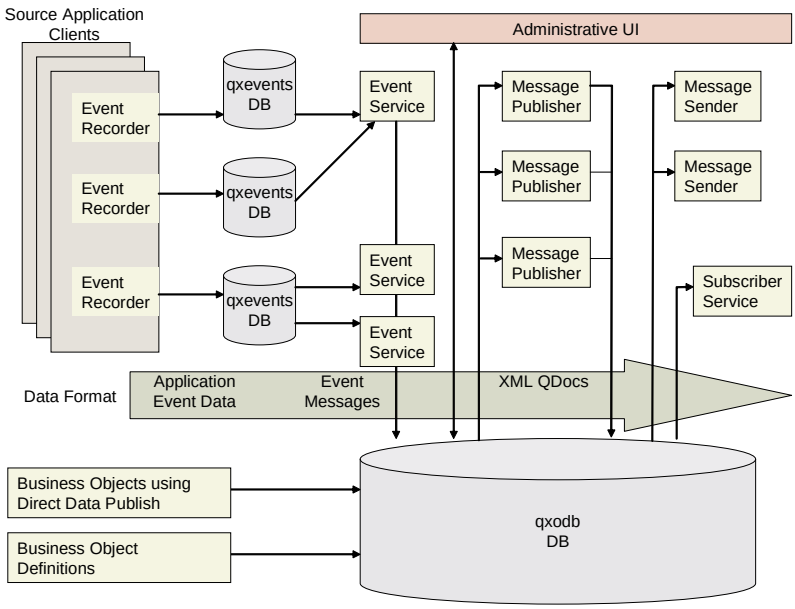
Several configuration steps are required in order to implement DDP in QAD Enterprise Applications EE and QXtend. For details, see Chapter 11, “QXtend Interaction with Financials,” on page 131.

Note DDP documents must be associated with a valid business event, otherwise the document will not be published. The business event is provided in the session context parameter. The system checks to verify that the business event is a valid event type for the source application. If the event is not valid, the system raises an exception. For details about session context parameters, see Appendix B, “Parameter Data,” on page 401.

The QXO Outbound Process

Creating QDocs from extracted QAD Enterprise Applications data is accomplished in several steps. Each step is defined in—and can be monitored in—the QXO administration interface.

Fig. 1.3
Basic QXO Architecture



The steps for data include the following:

1. A change occurs to watched data in QAD Enterprise Applications. Change notifications are written out to the `qxevents` database.

   Note DDP business objects are passed directly from the source application to the `qxodb` database in QXO, then processed by the message publisher.

2. QXO queries QAD Enterprise Applications for the changed records.

3. The extracted data is published as QDocs.

4. The QDocs are delivered to the assigned subscribers.

Taking each step in turn, the product architecture and implementation requirements become clear. Refer to Figure 1.3 on page 14 to see product architecture and data flow through QAD QXtend.

Change to Watched Data in QAD Enterprise Applications

Data becomes watched when you install and then activate replication triggers on the tables for a given business object. In order to avoid performance overhead, the triggers are simple: they write out a notification of a changed record to a side database, `qxevents`. A change is any add, modify, or delete to these tables.

The triggers required for supported QXO business objects are shipped and installed with the product. However, even unused triggers, if active, create a small performance overhead. Therefore, each implementation determines which triggers are activated.

Triggers are not required for DDP business objects since the entire business object is passed directly to QXO—it is not necessary to watch event data for these objects.

QXO Queries QAD Enterprise Applications

QXO uses an event service to connect to the `qxevents` database and poll for change notifications, using either the table rowid or a unique identifier field on the database table to identify where changes occurred. When it encounters a change record, the event service queries the QAD Enterprise Applications tables that make up the impacted business objects in QXO.

For business objects that employ DDP an event service is not required since these business objects are passed directly to the `qxodb` database in QXO, and from there processed as usual.

One table can be a component of multiple business objects. For example, the customer master, cm_mstr, is a component of the customer, customer item, and sales order business objects, among others. Depending on setup, queries can be for the full business object, or only for the primary index fields and any changed field data.

No query against QAD Enterprise Applications is run when:

- No valid or active business object exists for the changed data.
- No active profiles exist for the data.
- No active subscribers request this data.

The event service writes the extracted data to the QXO database.

Publishing the Data

Next, the message publisher generates one or more QDocs using the profiles as templates and stores the QDocs in `qxodb`. The QDocs generated are XML documents in proprietary QAD format. The SOAP-compliant container—an envelope, a header, and a footer—in which the QDoc is wrapped by the message sender (if the subscriber is thus configured) allows the QDoc to be sent to Web services that accept or poll for XML documents.

Delivering the Data

The QDocs can be delivered to a directory location where the subscriber polls for and picks up new QDocs. Alternately, the QDocs can be delivered to a Web service listening for new QDocs through a URL (Universal Resource Locator).

The subscriber can be another QAD Enterprise Applications instance, another QAD product, or a third-party application or messaging broker set up to receive and interpret QDocs. QDocs moving between QAD products start as outbound QDocs and are picked up by QAD QXtend Inbound.

Implementation Overview

This section provides an overview of the steps required to implement and customize QXO.

Implementing QXO

1. Configure each QAD Enterprise Applications instance as a source application.
2. Activate all required event types for each source application.
3. Configure QXO services: event services, message publishers, and message senders.
4. Set up subscribers.
5. Identify the required business objects.
6. Set up or modify the required business objects. For each business object you can:
   - Add or modify the table joins to use.
   - Add or modify the WHERE clause to select what data to retrieve.
   - Set fixed export values for specific data elements.
   - Write and assign calculations for specific data.
   - Add custom fields.
   - Determine if unchanged data is published.
7. Create profiles.
8. Activate the required triggers in QAD Enterprise Applications.
9. Define delivery schedules (optional).
10. Define e-mail alerts (optional).
11. Configure archive settings (optional).
12. Validate your configuration (optional).

Customizing QXO

You only need to customize QXO for QAD Enterprise Applications if you want to modify existing business objects or add new ones. More extensive customization is needed if you want to add business objects for custom QAD Enterprise Applications applications or a Progress-based application other than QAD Enterprise Applications.

To modify QAD Enterprise Applications business objects or add new ones, you must:

1. Define the business object requirements.
2. Identify the QAD Enterprise Applications tables required and activate or add these event types.
3. Update `qaddb` schema and compile supplied trigger files for any tables not currently configured for QXO.
4. Follow implementation steps 4 through 12 in the previous section.

For details, see “Creating New Business Objects” on page 76.

Chapter 2

Implementing QAD QXtend Outbound

The following material covers the steps required to configure QAD QXtend Outbound (QXO).

Introduction

QXO relies on a sequential processing flow to:

1 Gain notification of a change in a source application.

2 Extract data from the source application.

3 Publish the data as XML QDocs.

4 Send the QDocs to subscribers.

Note Steps 1 and 2 in the sequence above do not apply to business objects that employ DDP.

To implement this sequence:

1 Define your source applications.

2 Define an event service to check for and then extract the data.

3 Define a message publisher to publish the QDocs.

4 Define a message sender to send the QDocs.

5 Define subscribers to receive the QDocs.

6 Define a delivery schedule (optional).

7 Define e-mail alerts (optional).

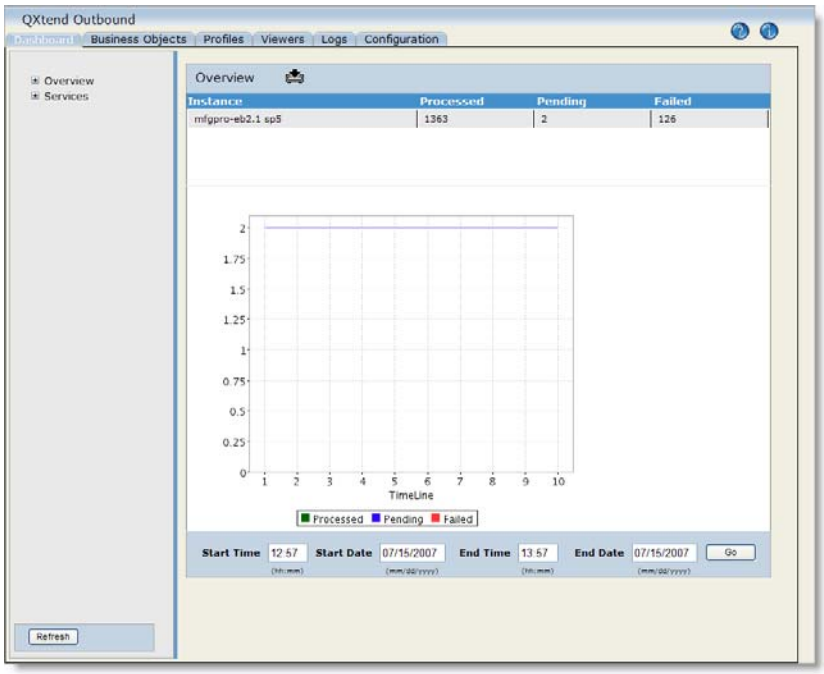
8 Configure archiving settings (optional).

Prior to implementing the QXO processing sequence, users unfamiliar with Web or HTML applications may want to review the following material. If you are comfortable using a Web-based application, skip to “QXO Source Applications” on page 23.

QXO Console

QXO displays as a multi-function console that includes graphing capabilities and lets you view, monitor, and configure QXO components through a set of tabs. Figure 2.1 illustrates the initial view displaying the Dashboard tab.

Fig. 2.1
QXO Administrative Console



The various tabs provide quick access to the following functions:

Dashboard. Display the current status and statistics about QXO processes and drill down to more detailed views.

Business Objects. Create or modify business objects in a tree view and display data related to each node in the object.

Profiles. Create or modify business object profiles in a tree view and review and modify the fields associated with each node.

Viewers. View messages generated by QXO at various points in the transformation process.

Logs. View logs related to events, messages, and QDocs.

Configuration. Create and update profiles for components of QXO, including source applications, event services, message publisher services, message sender services, delivery schedules, archive settings, and subscribers.

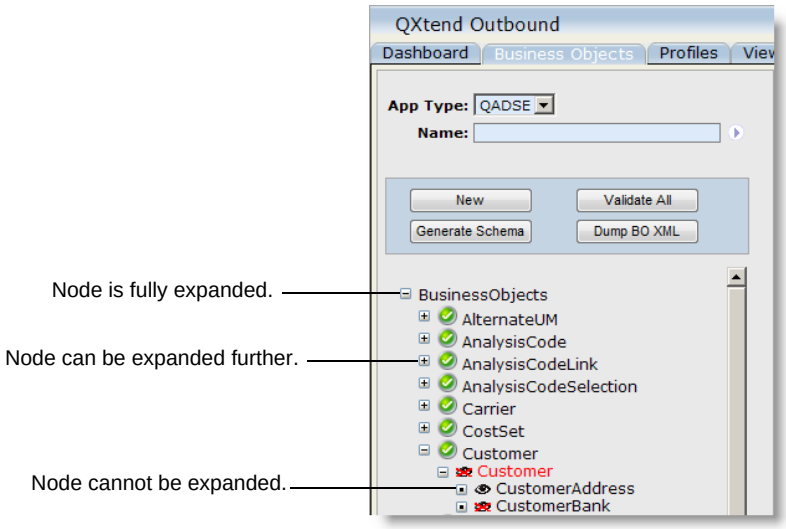
General UI Functions

The QXO administrative user interface uses tab-based HTML forms. This section discusses common aspects of the UI in the various functions, and describes how to use them.

Navigating Tree Views

Many of the QXO administrative screens display items in a tree view. Navigate the tree by clicking nodes to expand or collapse them. Icons indicate if a node is expandable, fully expanded, or cannot be expanded. Figure 2.2 illustrates a navigation tree.

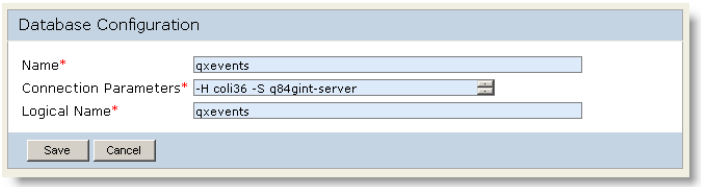
Fig. 2.2
Navigation Tree



Entering Data in HTML Forms

The data entry forms as illustrated in Figure 2.3 show field names and values. Access the fields by clicking or by tabbing through the fields. Fields displaying a red asterisk (*) are required.

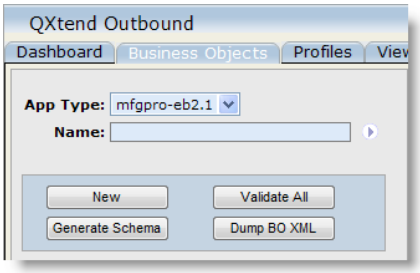
Fig. 2.3
HTML Data Entry Form



Filtering Names

On many QXO administrative screens, you can enter a name to filter the items displayed for update or viewing. Figure 2.4 illustrates the Name filter field for business objects. The rules for entering values in other Name fields are the same as for this one.

Fig. 2.4
Name Filter Field



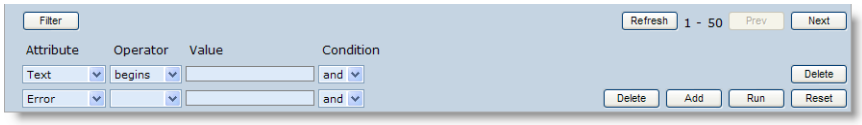
Enter a value in the Name field to determine what displays in the left pane for selecting items to view or update.

- Enter a specific search value to find an exact match. For example, enter Carrier to update the carrier record.
- Enter a search value followed by the wild card (*) to find all items with names that start with that value. For example, enter sales* to see business objects with names that start with sales.
- Leave blank to display all items.

Using Table Filters

On QXO screens where you review data, such as the logs and viewers, you can filter data by one or more sets of criteria. To add or modify filter criteria, click Filter. A screen like the one in Figure 2.5 displays.

Fig. 2.5
Filtering



Use the Add button to add a row for specifying an additional filter. Use the Delete button to remove a filter line. Click Run to apply the filter to the displayed information. Click Reset to close the filter view.

Choose an attribute from the drop-down list. The attributes match the columns of data being displayed. For example, to filter by message text, choose text as the attribute for the filter.

You can use the following operators with filters:

Operator	Description
Text Operators	
begins	The system returns records with values that start with the value you specify.
>	The system returns records with values that start with a value greater than the one you specify.
matches	The system returns records with values that exactly match the value you specify.
Numeric Operators	
>=	The system returns records with values that start with the value you specify or any value higher than it.
<	The system returns records with values that start with the a value less than the one you specify.
=	The system returns records with values that exactly match the value you specify.
<>	The system returns records with values that do not exactly match the value you specify.
<=	The system returns records with values that start with the value you specify or any value lower than it.

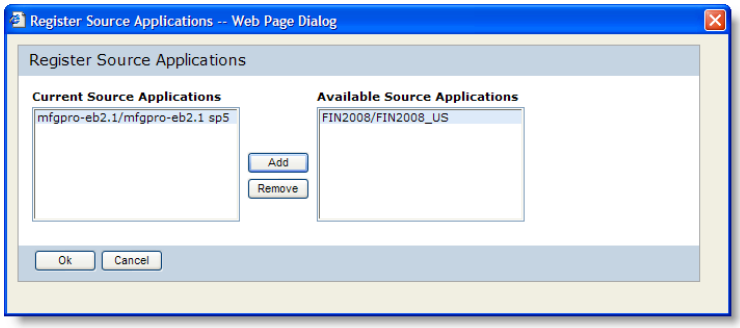
Use the Condition setting when you have more than one filter criterion to indicate whether you want values that meet all conditions to be returned (AND condition) or values that meet any one of the conditions (OR condition).

Using Lookups

In many QXO screens, you can associate one type of record with another. For example, you can associate source applications with event services. To make this kind of association, you choose values from a lookup that displays all available records in one list and the records currently associated in another.

These lookups all operate in the same fashion. The screen shown in Figure 2.6 illustrates the lookup for source applications.

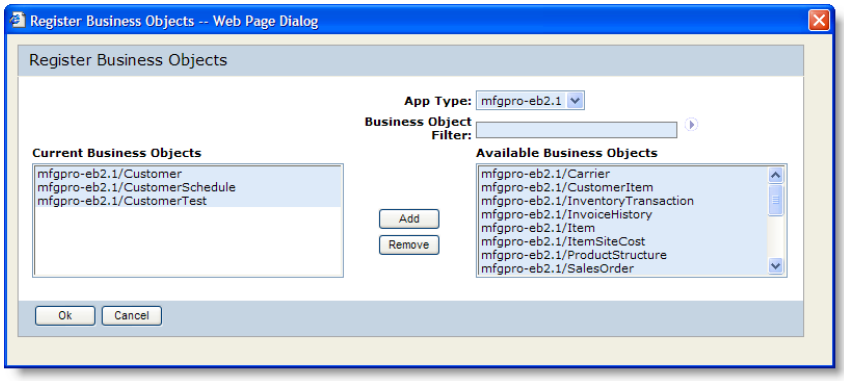
Fig. 2.6
Register Source Applications for a Session Profile



Use the Add and Remove buttons to move selected objects from one list to the other. Click OK when you are satisfied with your changes; click Cancel to discard changes without saving them.

In some lookups, you can filter the values that display in the available records list. For example, when you register business objects, additional fields display that let you filter the list, as shown in Figure 2.7.

Fig. 2.7
Register Business Objects for a Message Publisher



Enter a source application value in the App Type field. Then specify a filter value in the Business Object Filter field—or leave the field blank to display all. Click the arrow to display matching business objects in the Available Business Object list. Then use the Add and Remove buttons to move values from one list to the other as needed.

QXO Source Applications

Source applications are the specific databases from which the QXO event service extracts data. Each source application database that shares a unique schema—for example, all QAD Enterprise Applications eB2, SP3 databases—can be managed under a single source application type. For version eB2.1 or above databases, this also extends to multiple domains that share a schema definition.

A source application type also lets you identify the specific event types—database triggers and/or business events—you want to extract from the databases within that type. Event types are essentially the tables within the source databases.

Source applications that employ DDP post business objects directly to QXO via the service interface API, and consequently do not require an event service. For DDP source application types there is no need to define databases, domains, or business object groups.

You can also define business object groups for any source application. Business object groups enforce the correct chronological sequencing of data messages from dependent business objects. It is required, however, that all active event types of these business objects are all registered with the same event service. Each business object can belong to only one group.

See “Sequencing Business Dependencies” on page 11.

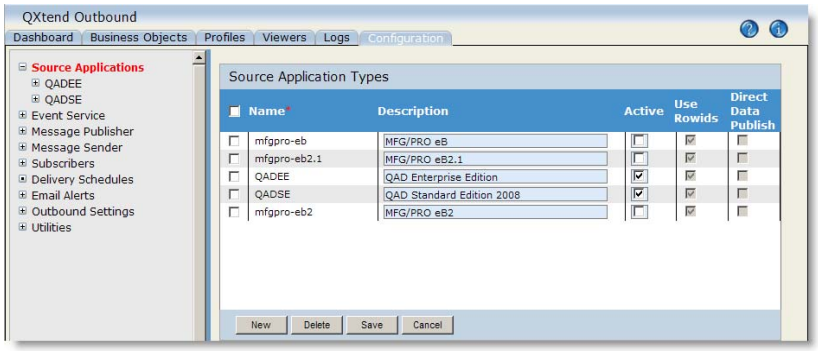
Note Business object groups cannot be defined for business objects that employ DDP.

In this first step, you define all the source applications from which you require data. You must also define a source application before you can update business objects or profiles.

In the QXO Console, click the Configuration tab to gain access to all the processing implementation services. Click the Source Applications node in the left-hand tree view. If it was not already displayed, the Source Application Types screen displays.

1 Click New in the Source Application Types screen. An entry line is added to the screen.

Fig. 2.8
Creating a New Source Application Type



Name. Specify the name of the new source application type.

Description. Enter a description as required.

Active. Select this option to indicate the source application type is active. The source application type must be active to be available on the Business Object and Profile tabs. In addition, the event service will not extract data from a source application unless the source application is active.

Use Rowids. This setting indicates whether the source application type uses a rowid or object identifier (OID) to identify the data object that has changed. A rowid is an internal system-generated identifier used by the database, whereas an OID is an actual field on the table.

Source application types eB and eB2 use rowids only. Types eB2.1 and above can use either a rowid or OID.

Note In order to use the OID index in eB2.1 or QAD SE you must first load the optional Enhanced Controls schema. For the QAD EE source application loading this schema is not required since by default the OID index is on.

The Use Rowids option is preset and cannot be changed for predefined source application types. Source applications that do not use rowids require the unique identifier in the data object to be specified. For details see “Modifying Business Object Data Objects” on page 80.

Direct Data Publish. Indicates if this source application posts event data directly to QXO. This setting cannot be changed for predefined source application types.

If this setting is selected, the source application node on the tree menu does not display further information—databases, domains, business object groups, and event types, for example.

For details see “Direct Data Publish” on page 13.

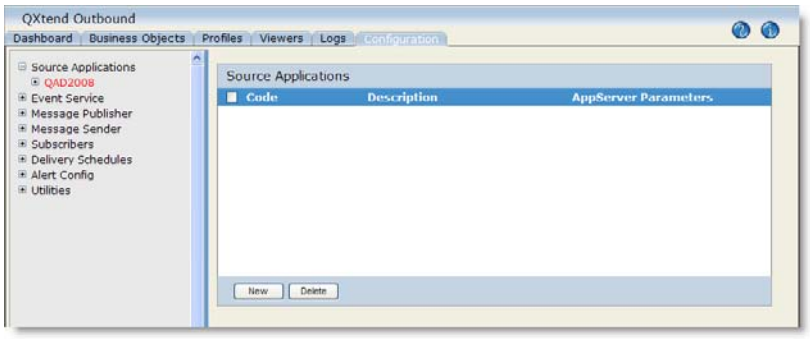
Important In order to load the QAD standard business objects and profiles, the business objects use the following predefined source application types provided with QAD QXtend:

- mfgpro-eb
- mfgpro-eb2
- mfgpro-eb2.1
- QADSE
- QADEE

If you choose to change the naming convention, the folders containing the QAD-standard business object XML to be loaded must be renamed to the defined source application type. See “Importing Business Objects and Profiles” on page 56. The XML resides in `QXOsrvInstallDir\boXML`.

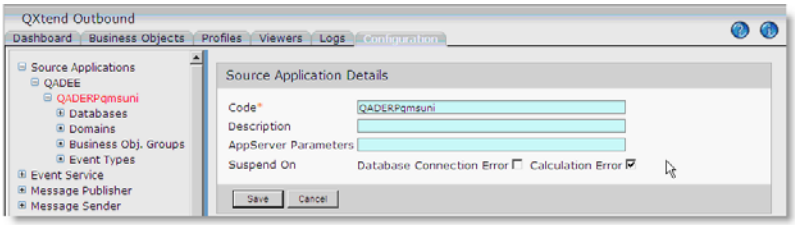
2 Click Source Applications in the navigation tree to open the node. The active source application types display. Click it to open the Source Applications screen.

Fig. 2.9
Source Applications



3 Click New to display the Source Application Details screen.

Fig. 2.10
Source Application Details



Code. Enter a code identifying a source application from which QXO messages originate. You use this code to identify the source application in the event services definition.

Description. Enter a brief description as needed.

AppServer Parameters. In general, enter the NameServer application service (`-AppService`) and the AppServer host (`-H`); for example:

```
-AppService qxoappservice -H co9906
```

In the example, the application service is `qxoappservice` and the host is `co9906`.

The AppServer Parameters field supports the use of calculated field programs documented in “Implementing Calculated Fields” on page 63.

Note If calculated field programs are not required to be executed on the AppServer, then this field can be left blank.

Additional values can be entered in this field. For details on AppServer parameters, see the Progress documentation.

Suspend on. Specify whether to suspend the source application on one or both of the following errors: database connection error and calculation error.

If you select Database Connection Error, the source application will be automatically suspended on database connection error; if you select Calculation Error, the source application will be automatically suspended on calculated field program 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.

You can manually resume or suspend source applications through QXO Dashboard or batch process scripts. Batch process query shows the Suspended status.

When a source application is resumed, QXO services automatically reprocesses the extracted data and raw messages with the ERR status.

You can configure e-mail alerts to be sent on calculation and database errors. See “QXO E-mail Alerts” on page 50.

4 Click Save to save the new source application.

5 Repeat these steps for each source application in the source application type.

Adding Source Application Databases

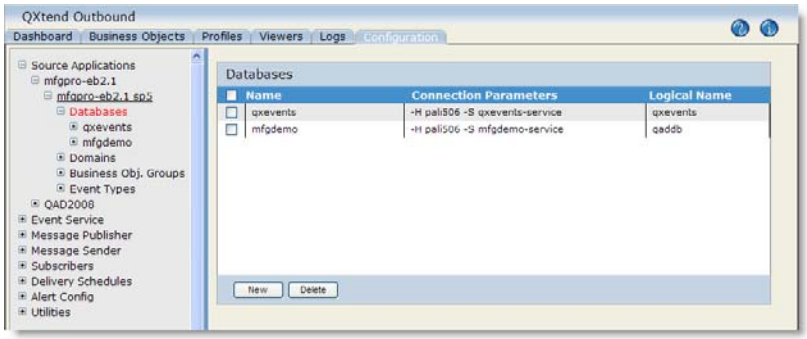
Each source application can contain multiple databases, as long as all the databases in the source application type share the same schema.

Note Source application databases are not relevant for source application types that employ DDP.

1 Click the source application in the navigation tree under the source application type. The tree expands to display the source application components.

2 Click Databases under the source application. The Databases overview screen displays.

Fig. 2.11
Databases



3 Click New to add a database. You must define the production `qaddb` databases as well as the `qxevents` database.

Fig. 2.12
Database Configuration

Name. Specify the physical file name of the source application database (without the `.db` extension).

Connection Parameters. Specify any Progress connection parameters the system should use to connect to this database. For details, see the *Progress System Administration Guide*.

Logical Name. The `qxevents` database must have the logical DB name `qxevents`.

4 Click Save to save the database definition.

5 Repeat these steps for each database in the source application.

Databases are disconnected whenever you modify the database configuration. By default, the database is automatically reconnected when you have finished the modifications and choose Save. If you encounter reconnection errors, refer to *Installation Guide: QAD QXtend* for assistance.

Adding Source Application Domains

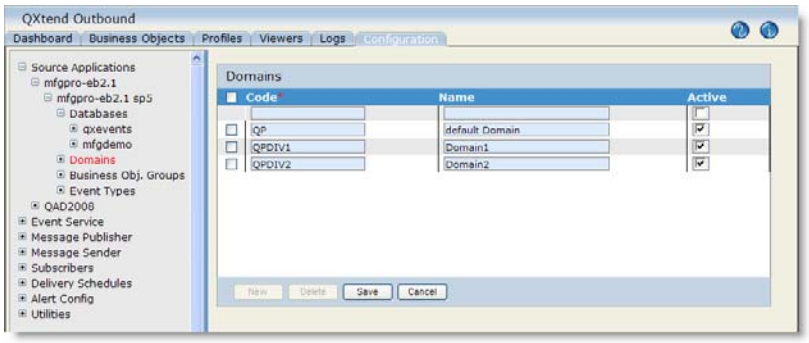
Domains exist in the QAD Enterprise Applications databases for versions eB2.1 and above. If no domain is defined here for the source database, then data from all domains in the source database is extracted. If subscriber applications require the domain designation of extracted data, you must add one or more domains to the source application. Otherwise, domains are optional.

Note Domains are not relevant for source application types that employ DDP.

1 Click Domains under the source application. The Domains overview screen displays.

2 Click New to add a domain.

Fig. 2.13
Domains



Code. Enter a code identifying a domain in the source application.

Description. Enter a brief description.

Active. Indicate if this domain is active. If this is not checked, future data processing that accesses this domain will not occur.

3 Click Save to save the domain definition.

4 Repeat these steps for each domain in the source application.

Adding Source Application Business Object Groups

The advantage of creating a business object group is that when all active event types within the group are registered with the same event service, the tables within the group are processed in the sequence you set up. For instance, you may always want a customer's base data (ad_mstr and cm_mstr) processed prior to sales data (so_mstr and sod_det).

If active event types in a business object group are not registered with the same event service, business objects in the group will be processed by different event services, which defeats the purpose of business object groups since you cannot guarantee that data of designated business objects will be processed as set up in the group in a single-thread manner. This gives rise to a warning message in configuration validation.

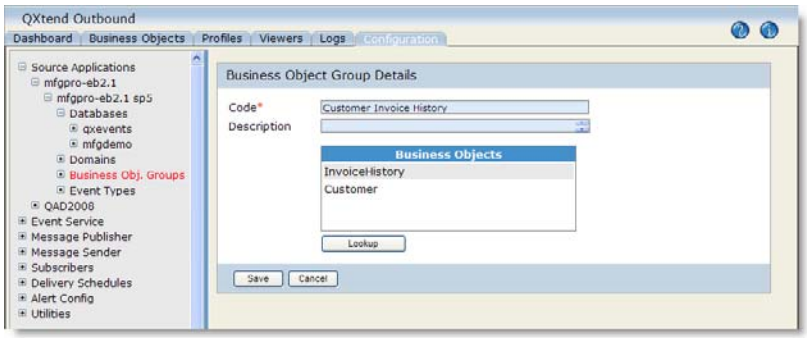
Note Due to sequencing requirements, these business objects are processed in a single thread. Since this will inhibit performance, business object groups should be used only if absolutely necessary.

Note Business object groups are not relevant for source application types that employ DDP.

1 Click Business Object Groups under the source application. The Business Object Groups overview screen displays.

2 Click New to add a new business object group. The Business Object Group Details screen displays.

Fig. 2.14
Business Object Group Details

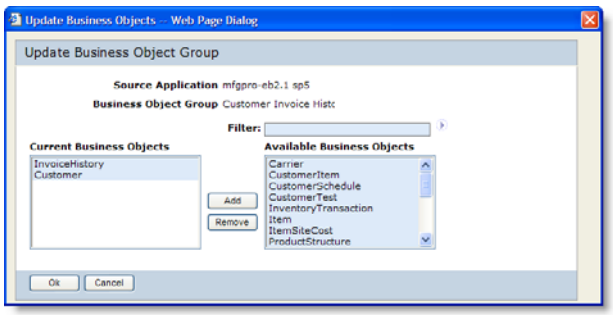


Code. Enter a name for the business object group.

Description. Enter a brief description.

3 Click Lookup to add the business objects to the group. The Update Business Object Group lookup displays, listing available objects.

Fig. 2.15
Update Business Object Group Lookup



4 Optionally, you can enter a filter such as `cust*` to reduce the number of business objects you see. Click the arrow to the right of the filter box to display either filtered or unfiltered business objects.

5 Select business objects from the Available list and click Add. To remove groups, highlight the group name in the Current Business Objects list and click Remove.

6 Click OK to save the business objects.

7 Click Save to save the new business object group.

Editing Source Application Event Types

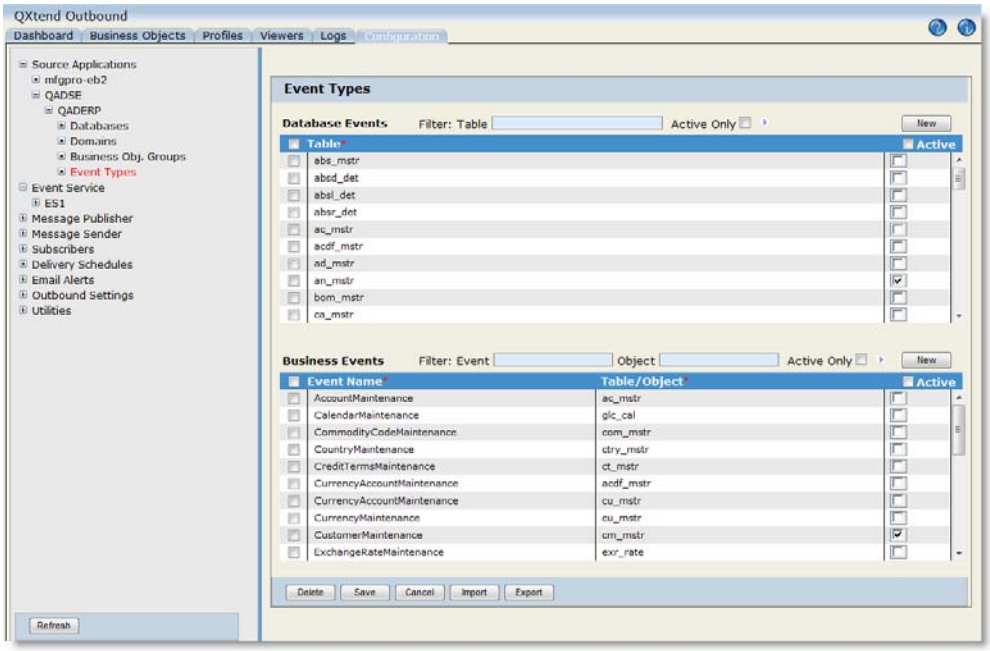
Source application event types—either database triggers or business events—are the tables that are activated for this source application. Before any data can flow through QXO, you must manually select which event types to turn on for each source application you define. The event types are not active by default because there is a performance impact associated with the lookups QXO must do for each event.

You can edit the available events to make particular tables inactive for this source application. This can be accomplished under the event types node under the appropriate source application.

If you deactivate a table, the trigger for that table in the source application still executes, but no record is written.

1 Click Event Types under the source application. The Event Types overview screen displays.

Fig. 2.16
Event Types



2 Select the Active check box to activate a data object; clear the Active check box to deactivate a data object. The Active check box indicates if data should be extracted for this data object. If not, the triggers still execute in QAD Enterprise Applications, but no data is extracted.

3 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:

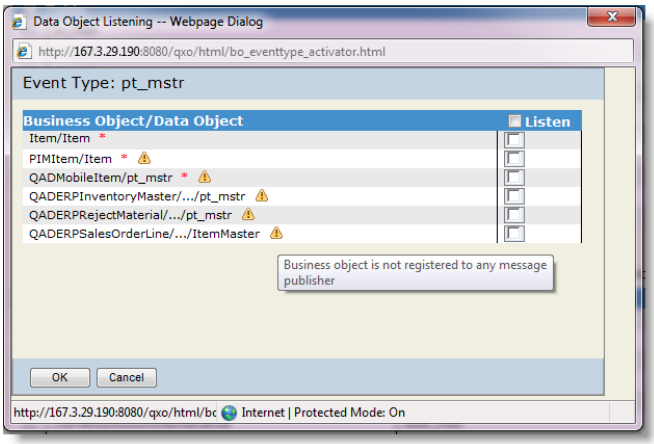
- *BusinessObject*/*DataObject* * for data objects at the top level of business objects with the asterisk indicating that it is a top-level data object
- *BusinessObject*/.../*DataObject* for data objects at lower levels of business objects. To see the full path of a data object under the business object,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

A warning symbol displays if a business object is not registered to any message publisher.

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

Important 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 register with any event service to be processed.

Fig. 2.17
Data Object Listening Configuration



4 Click Save to save your changes.

Important If you navigate to other screens without saving, any changes you made to the event type and data object configurations before your last save will be lost.

Adding Source Application Event Types

The business objects supplied with QXO use only a subset of QAD Enterprise Applications tables, which correspond to the set of default event types to be activated. If you create new business objects to accommodate your own business requirements that reference data from tables not already defined as events, you must add new event types and activate them. See “Creating New Business Objects” on page 76.

You will also need to do this if your business object includes custom tables or references tables in a non-QAD Enterprise Applications database.

Note Additional steps are required to use new event types with business objects. These are described in “Creating New Business Objects” on page 76.

You can add two types of events: database triggers or business events. You also can define business events for source application types that are DDP.

Click Event Types under the source application. The Event Types overview screen displays.

Adding Database Triggers

1 Click New.

2 A Database Event Type window is displayed. Specify a table name for the database trigger event type. The table name you enter must match exactly the name of the table in the application schema that it is associated with.

Fig. 2.18
Adding a New Database Trigger Event Type

3 Click Save to save your changes.

Note A warning message displays if the table does not exist. A warning symbol displays if the replication-write and/or replication-delete trigger is missing from the schema definition in the source application. A tooltip on this icon displays indicating if one or both of the triggers are missing.

Note A warning symbol displays if the event type is active but not registered with any event service.

Adding Business Events

Along with database triggers, business events can be added. This allows for business processes to trigger an event, rather than a database change.

You can add a business event manually by using the Add button and specifying the business event name, or use the Import button to import business events from the source application database.

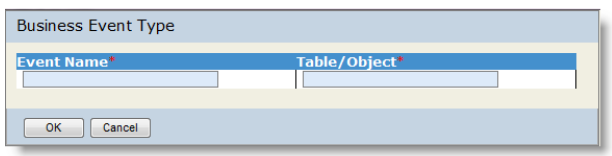
If you define business events manually, when you attempt to save the event type the system checks the source application database to ensure that the table or object you specify exists. If the system cannot find a table or object of that name, it checks the database for a business object. If the system cannot validate the business event entry, it displays an error.

To manually add a business event:

1 Click New.

2 A Business Event Type window is displayed. Specify the event name and associated table/object. The table/object name must be a valid table or a DDP business object.

Fig. 2.19
Adding a New Database Trigger Event Type

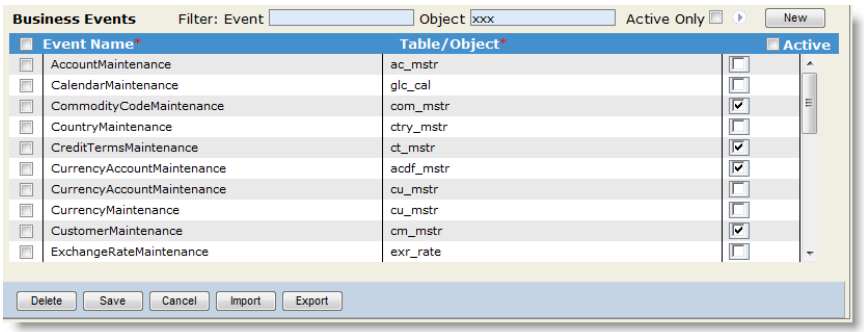


3 Select the Active check box to extract data for this event type.

4 Click Save to save your changes.

Business events can be exported by clicking the Export button. Business events are exported to the source application type folder in the `events` folder. For example, events for the QADEE source application type are exported to `qadapps/qxtend/qxoAppserver/events/QADEE`.

Fig. 2.20
Adding a New Business Event



QXO automatically detects business events in DDP published data and saves them as inactivated if the events are not in the QXO database.

Note This does not apply to DDP Only source applications.

In order to be published, DDP documents must be associated with a valid business event. The system checks to verify that the business event is a valid event type for the source application and raises an exception if the event is invalid.

Searching for Database and Business Events

You can use the Filter panel on top of the database and business events lists to search for specific event types.

To search for database events, enter a table name using a wildcard (for example, *_mstr), and specify the active status; then click the Search icon.

To search for business events, enter an event, table, object name, or a combination of them using wildcards, and specify the active status; then click the Search icon. For example, enter “*/bcurrency” in the Object field to find all event types for object bcurrency; enter “create/*” in the Event field to find all objects with the “create” event type.

Search/filter criteria is not case-sensitive.

QXO Services

QXO uses five types of services: event, message publisher, message sender, archive, and e-mail. The profiles for the event, message publisher, and message sender services contain the same parameters, the only difference being that:

- Event services are associated with source applications—except source applications that employ DDP.

- Message publisher services are associated with business objects.
- Message sender services are associated with subscribers.

You choose the source application, business object, or subscriber using a lookup that displays available values and lets you move names from one list to another.

You do not have to create the archive service or the e-mail service. The archive service is created automatically the first time the archive routine is run. For details, see “Data Archive” on page 57.

For information about viewing specific services and operating services from the command line, see “Viewing Specific Services” on page 99.

QXO Event Service Process

Event services are the QXO processes that initially connect to the `qxevents` database to identify what changes have occurred in the source application and then to the production database to extract the data and write it to the `qxodb`.

The actual process an event service follows is:

1. Poll for and identify an eligible event in `qxevents`.
2. Identify affected business objects in QXO.
3. Build a data object to hold the data.
4. Extract the data from `qaddb` into the data object.
5. Store the data object in `qxodb`.

Note These steps are not applicable to business objects that employ DDP since these business objects are passed directly to the `qxodb` database in QXO.

An event service can extract data from source applications using either the table rowid or an OID on the database table. The event service does not extract data from a source application if the source application type is not active. For details see “QXO Source Applications” on page 23.

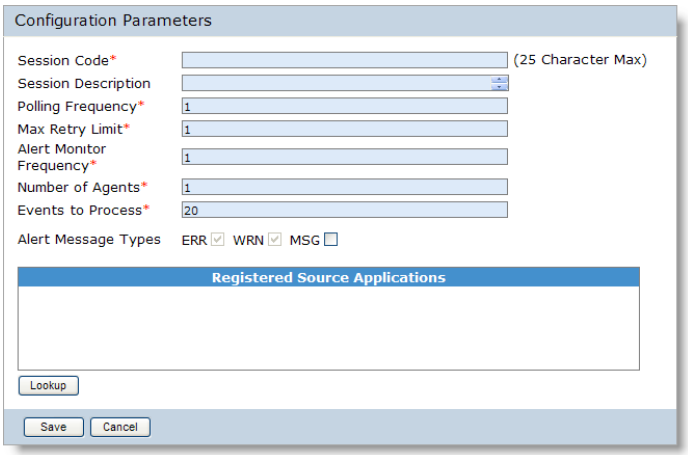
Note In QAD QXtend version 1.4, the XML format of event messages changed. Messages in the old format that were previously extracted and stored are converted to the new format when they are required.

Creating an Event Service

Event services are defined to operate with a specified set of source applications.

1. In the Configuration tab, click Event Service in the navigation tree. The Event Services overview screen displays.
2. Click New to create a new event service.

Fig. 2.21
Event Services Configuration Parameters



Session Code. Enter a code (maximum 10 characters) for this service. Using a naming convention such as EVS<*serviceName*> makes it easier to identify the service as an event service later on.

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

Polling Frequency. Enter the number of seconds for the service to wait between polls to the `qxevents` database.

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. See “Alert Message Types” on page 34.

Number of Agents. Specify the number of agents to start for the service when the AppServer is started.

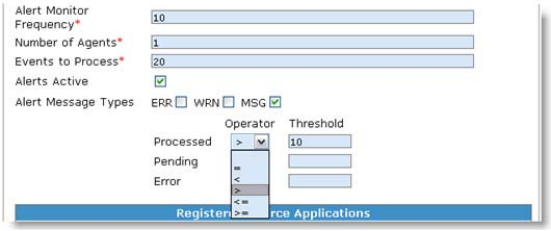
Events to Process. Enter the maximum number of events to be processed by a source application before switching to a different one. This is used only when an event service is defined with more than one source application, and is used to ensure throughput across all registered source applications.

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).

Note For details on configuring e-mail alerts for metric-related alert messages see “QXO E-mail Alerts” on page 50.

When you select MSG, additional options display.

Fig. 2.22
Setting MSG Alert Metrics



The MSG alert metrics let you specify alert thresholds for Processed, Pending, or Error conditions. In Figure 2.19, a threshold on processed messages is set. If the defined threshold is reached, an 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 were still more than 20 messages pending, another alert would be raised.

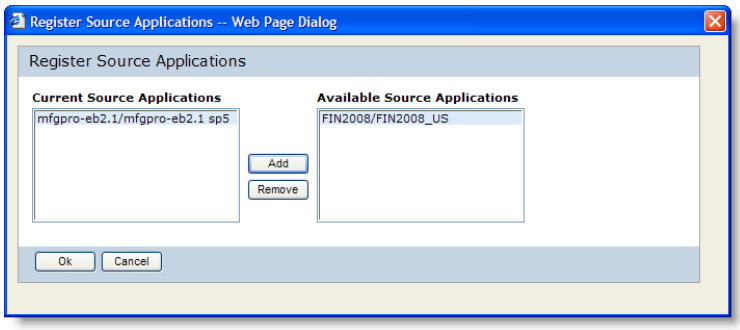
Note The type of conditions reflect the type of service being configured. For an event service, for example, the conditions relate to event processing. For the message publisher and message sender, the conditions relate to QDoc creation and QDoc delivery, respectively.

Operator. Choose the numeric operator for each processing condition.

Threshold. Enter the number of events to occur before an alert is issued.

3 Click Lookup to register source applications with this service. A lookup displays available and currently assigned source applications. Use the Add and Remove buttons to update the lists.

Fig. 2.23
Source Application Lookup



4 Click OK to save and exit the Register Source Applications screen.

5 Click Save to save the event service configuration.

Registering Event Types with an Event Service

After you create a new event service, it appears under Event Service in the navigation tree. By default, all active event types in the event service's registered source applications are automatically registered with it as well. However, you can configure an event service to only

process specific event types so as to create dedicated event service instances for high-priority events, allowing them to be processed as soon as they arise. This is achieved through registering specific event types of a source application with an event service for processing.

1 Under Event Service in the navigation tree, click the event service you want to configure to display all its registered source applications under it.

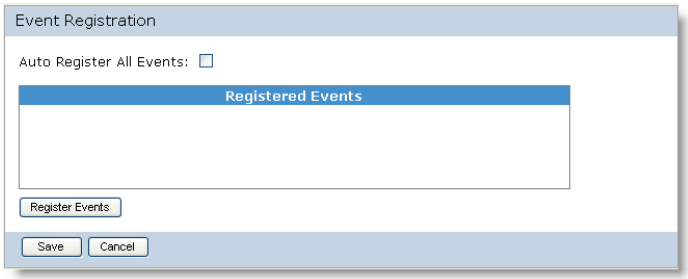
2 Click a source application.

3 The Event Registration screen on the right shows that all active event types in the source application are automatically registered with the event service. Clear the Auto Register All Events option.

Fig. 2.24
Event Registration (Default)

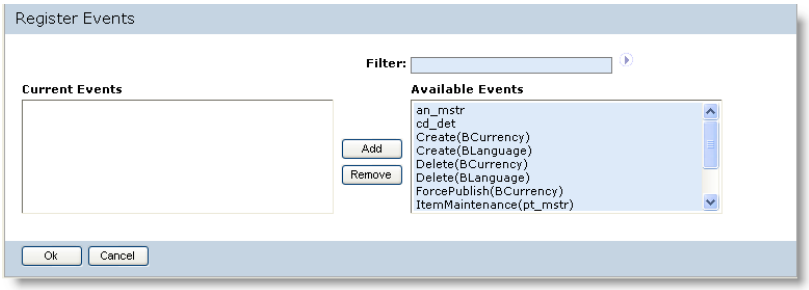
4 Click Register Events to register event types with this service.

Fig. 2.25
Event Registration



5 A lookup window displays available and currently registered event types. Use the Add and Remove buttons to update the lists.

Fig. 2.26
Event Registration Lookup



6 Click OK to save and exit the Register Events screen.

7 Click Save to save the event service configuration.

Defining a Message Publisher

Message publishers poll the `qxodb` database for new data objects created by an event service, as well as for new business objects passed directly to the `qxodb` database from the QAD EE Financials module.

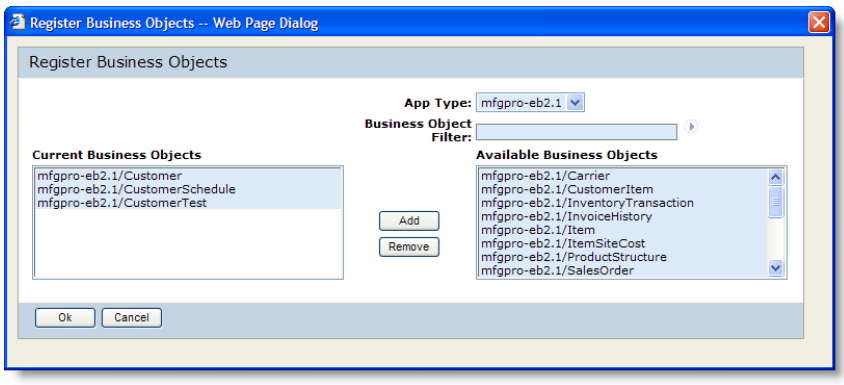
Message publisher services:

1. Poll for and identify an eligible data object in `qxodb`.
2. Identify affected business objects in QXO.
3. Identify applicable profiles.
4. Build an XML QDoc using a profile.
5. Store the QDoc in `qxodb`.
6. Create a delivery request for each associated subscriber, using the appropriate QDoc syntax for each subscriber.
7. Store the delivery requests in the subscriber's delivery queue.

As described in "QXO Services" on page 32, the parameters for message publishers are identical to those for event services.

1. In the Configuration tab, click Message Publisher in the navigation tree. The Message Publishers overview screen displays.
2. Click New to create a new message publisher. Enter values as described in "QXO Event Service Process" on page 33.
3. Click Lookup to register business objects with this publisher. A lookup displays available and currently assigned business objects. Use the Add and Remove buttons to update the lists.

Fig. 2.27
Business Object Lookup



4. Click OK to save and exit the Register Business Objects screen.
5. Click Save to save the message publisher configuration.

Defining a Message Sender

Message senders poll the `qxodb` database for new QDocs created by a message publisher. Message sender services:

1. Poll for delivery requests in the subscriber's delivery queue.
2. Send QDocs—using the appropriate QDoc standard—to the designated subscribers.

Message senders can forward the QDoc either through a Web service call to a URL, or by dropping the QDoc into a directory. As described in "QXO Services" on page 32, the parameters for message senders are identical to those for event services.

1. In the Configuration tab, click Message Senders in the navigation tree. The Message Senders overview screen displays.
2. Click New to create a new message sender. Enter values as described in step 2 in "QXO Event Service Process" on page 33.
3. Click Lookup to register subscribers with this sender. A lookup displays available and currently assigned subscribers. Use the Add and Remove buttons to update the lists.
4. Click OK to save and exit the Register Subscribers screen.
5. Click Save to save the message sender configuration.

QXO Subscribers

A subscriber is a delivery point for a QDoc or other published XML document. Another application can be the delivery point, or an instance of QXI. Subscribers have an associated delivery schedule and delivery method: either a directory location or a Web service. The subscriber is responsible for retrieving the outbound document.

A subscriber can be defined using either the 1.1 or 1.0 QDoc syntax specification. In addition you can specify parameters that are used in the most common forms of SOAP headers—or define your own header if QXI is not the intended target application.

Note If your current QXtend installation is updated, all existing subscribers are converted to use the QDoc 1.1 specification standard. For details, refer to *Installation Guide: QAD QXtend*.

To bring a new subscriber up to date, a Force Publish function is available. You can also restrict specific source applications to ensure a subscriber sees only the data intended for them.

You also can configure profiles for individual subscribers by using the Subscriber Profile Configuration Parameters screen. Configuring a profile allows you to specify the event types you want to use for that subscriber profile. For details, see "Configuring Subscriber Profiles" on page 45.

Subscriber E-mail Notification

Based on the result of processing subscriber messages, the system can send e-mail to a user that generated the event (for non-DDP source applications) or to the user defined within the message structure (for DDP applications).

When an event is generated by a non-DDP source application, the system stores the name of the user that triggered the event. When the event service successfully extracts the raw message, the system determines the email address of the user responsible for the change.

Messages that are sent via DDP can define email addresses of recipients interested in the result of the subscriber message. If a user address is included in the session context, it will always send an e-mail to the addresses in the session context, overriding the Email Data Owner setting in the subscriber configuration.

Raw messages generated by the event service are delivered using the subscriber settings. Messages that are sent via DDP use session context values. Delivered e-mail messages are formatted using the current default email template. For details on defining custom e-mail configurations, see “Managing Alert Groups” on page 52.

Free Use of QXO

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.

Table 2.1 describes the free use of QXO and QXI.

Table 2.1
Free Use of QXO and QXI

Communication Method	Subscriber Type	Outbound Free	Inbound Free
File Drop	Ext. application	No	n/a
File Drop	QAD application	Yes	n/a
Web Service	Ext. application	No	No
Web Service	QAD application	Yes	No
QXtend Web Service	QXtend	Yes	Yes
QXtend Web Service	QAD application	Yes	Yes

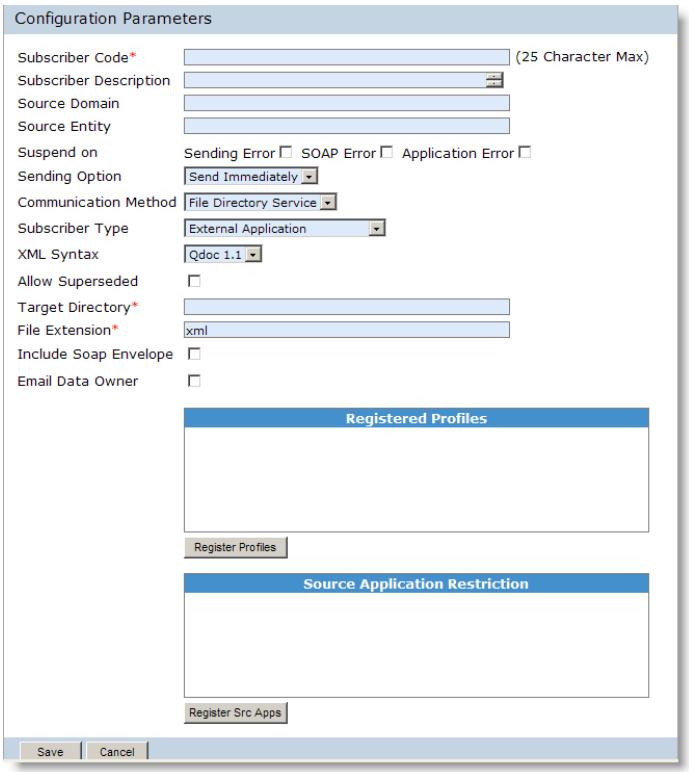
Note You can use the QXtend Inbound Usage Report to view which messages originated from licensed applications and which are free QAD messages. For details see “QXtend Inbound Usage Report” on page 262.

Defining a Subscriber

To define a subscriber:

1. In the Configuration tab, click Subscribers in the navigation tree. The Subscribers overview screen displays.
2. Click New to create a new subscriber.

Fig. 2.28
Subscriber Configuration Parameters



Subscriber Code. Enter a code (up to 15 characters) that identifies the external system or application.

Subscriber Description. Enter a description (up to 35 characters).

Source Domain. Identify any domain this subscriber requires (for version eB2.1 and higher source applications only).

Source Entity. Identify any entity this subscriber requires (for version eB2.1 and higher source applications only).

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.

You can configure e-mail alerts to be sent on calculation and database errors. See “QXO E-mail Alerts” on page 50.

Sending Option. Choose a sending option to be used when sending messages.

Send Immediately. Send the document to the subscriber immediately. This is the default if there are no delivery schedules defined.

Use Delivery Schedule. Send the document to the subscriber using a defined delivery schedule.

Delivery Schedule. This drop-down list displays only when the Use Delivery Schedule option is selected. The default delivery schedule displays as selected in the Delivery Schedules summary screen. Choose a different delivery schedule if required. See “QXO Delivery Schedules” on page 47.

Communication Method. Choose the method used to send messages.

File Directory Service: In this method, outbound QDocs are placed in a directory for the subscribing application to read. Specify a target directory. Selecting this option causes the subscriber Type to default to External Application.

Web Service: In this method, outbound QDocs are delivered to a URL where the subscriber is set up to listen for incoming documents. Specify a target URL. Selecting this option causes the subscriber Type to default to External Application.

For the fields specific to a Web Service subscriber, see Figure 2.29 and the field descriptions following it.

QXtend Web Service: In this method, users can connect to QAD QXtend without needing to know the full URL of the Web service. Selecting this option causes the subscriber Type to default to QAD QXtend.

The QXtend Web Service method also denotes communication that is carried free of charge.

Note For details about QAD QXtend licensing, see Chapter 22, “Licensing,” on page 255.

For the fields specific to a QXtend Web Service subscriber, see Figure 2.30 and the field descriptions following it.

QAD Workflow: The system sends messages through the QAD Workflow Alerts Event Publishing Web service. For details, see *User Guide: QAD Workflow Alerts*.

Subscriber Type. Specify the type for the subscriber. The options on this dropdown represent the valid applications that have been registered with QXI. For details about the communication method/subscriber type combinations that permit the use of free messaging, see Table 2.1 on page 39.

Note If the communication method is File Directory Service or Web Service, the default type is External Application. If the method is QXtend Web Service, the default type is QAD QXtend.

XML Syntax. Specify the version of the XML syntax to use for the subscriber. The options are QDoc 1.1, QDoc 1.0, or Other. Selecting the Other option requires the SOAP action—as well as SOAP envelope and header—to be configured.

Allow Superseded. Indicate whether the subscriber only accepts the latest revision of a document or incremental updates of a document. This option is typically used when multiple QDocs exist for the same rowid or OID that has been updated multiple times, and only the latest QDoc is required.

Yes. The subscriber only accepts the final revision of a document. Setting the Allow Superseded option to Yes means messages containing errors that were not sent and were superseded are marked as such. These superseded messages cannot be resubmitted.

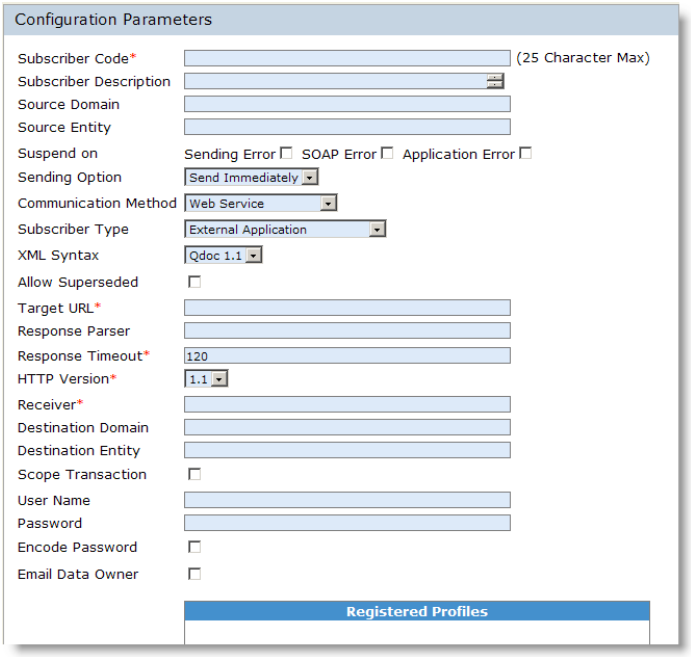
No. The subscriber only accepts incremental updates of a document. Setting the Allow Superseded option to No means that when messages containing errors are superseded, further updates cannot be sent until the errors in the message have been fixed and are set to the HOLD status until the previous message is successfully sent or deleted. If there are two messages with

the same rowid or OID for the same subscriber, messages are sent in a single thread in order to prevent the most recent document from being sent to the database first. All errors in the message must be fixed before further updates can be sent—or the QDoc can simply be deleted.

Target Directory. For file directory service, specify the location on the file system where outbound QDocs for this subscriber should be placed. If a full path is not specified, QDocs are placed in the current working directory.

File Extension. For the file directory service, designate the file extension of QDocs for filedrop subscribers; for example, you can specify `.req` as the QDoc file extension so that QDocs can be directly dropped into a queue to be processed by Queue Manager.

Fig. 2.29
Web Service URL Subscriber Entry



Target URL. For Web service, specify the URL where outbound QDocs for this subscriber are sent. This URL must be set up as a Web service.

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. For information on developing your custom response parser, see Appendix C, “Response Parser,” on page 407.

Response Timeout. Enter a number of seconds for QXO to wait for a response from the subscriber. If no response is received in that time period, an error is raised for the QDoc in QXO.

HTTP Version. Specify which version of HTTP to use when making the Web Service call. Web services require SOAP headers on all incoming communications in order to validate the sender and destination. QXO supports HTTP versions 1.0 or 1.1. QAD recommends that you use HTTP 1.1.

Receiver. Specify the receiver associated with this subscriber. A receiver is a named QAD Enterprise Applications instance. This field is required if XML syntax version 1.1 or 1.0 is selected.

Destination Domain. Optionally, specify a domain in the receiver. Domains exist in the QAD Enterprise Applications databases for versions eB2.1 and above. This field displays only when XML syntax version 1.1 or 1.0 is selected.

Destination Entity. Optionally, specify an entity in the receiver. Entities exist in the QAD Enterprise Applications databases for versions eB2.1 and above. This field displays only when XML syntax version 1.1 or 1.0 is selected.

Scope Transaction. Select this option to indicate whether QDocs sent to this subscriber should be scoped as single transactions. This field displays only when XML syntax version 1.1 or 1.0 is selected. For details see page 287.

User Name. Specify a user name for logging in to QAD Enterprise Applications. If a user name is not provided, no user name information is entered into the context section of the QDoc. This field displays only when XML syntax version 1.1 or 1.0 is selected.

Password. Specify a password for logging in to QAD Enterprise Applications. Any password entered into this field is masked. This field displays only when either the XML syntax version 1.1 or 1.0 is selected. This field is disabled if the User Name field is blank.

Note For security, any user name and password login credentials specified here are masked as asterisk characters (*) when returned in the QXI response document.

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

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. Typically this check box is used in a data synchronization scenario when data in one domain may not exist in the target domain. You must select this check box to display the Message Status options.

Note This field is disabled if the Active check box in the Email Configuration Parameters screen is not selected; for details, see “Configuring E-mail Settings” on page 50.

Message Status. Select the processing result that will trigger the e-mail to be sent. (Message Status is available only when the Email Data Owner check box is selected.)

Include SOAP Action. If using the XML Syntax option of Other, select this option to include the SOAP action specified in the SOAP Action field. This option must be selected if the subscriber is for a QXI instance.

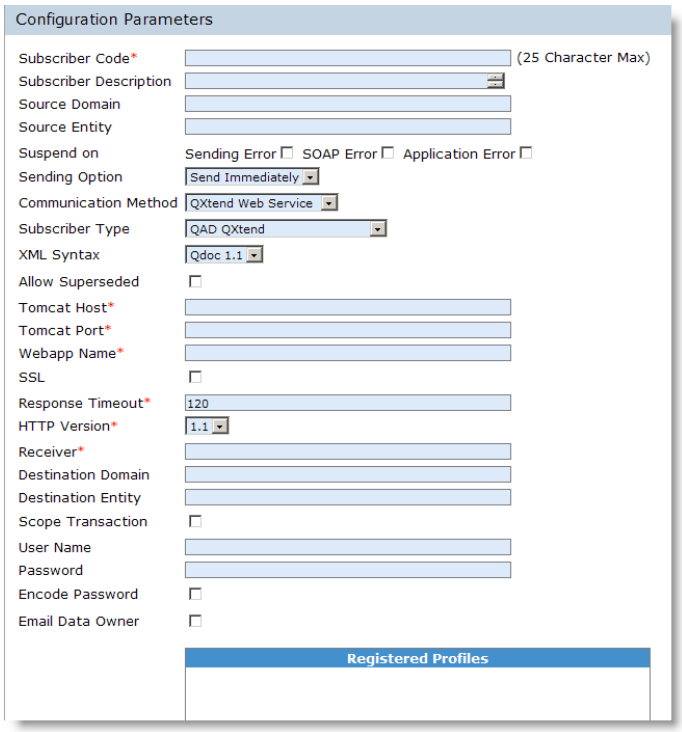
SOAP Action. If using the XML Syntax of Other with a SOAP action, enter a SOAP action, if required. Leave this field blank if the subscriber is a QXI instance.

SOAP Envelope. The SOAP envelope element is the root element of a SOAP message and allows QDocs to be sent to Web services that accept or poll for XML documents. When the message is sent, the &2 element of the SOAP envelope is substituted for the profile message. This option displays only when the Other option is selected on the XML Syntax drop-down. Use the default that displays in the text box.

SOAP Header. Copy the SOAP header from a valid QDoc directed to this subscriber and paste it in here. The SOAP header provides sender and destination details. This option displays only when the Other option is selected on the XML Syntax drop-down.

If you require a specific user-defined SOAP header for a specific Web service requirement, enter that header here.

Fig. 2.30
QXtend Web Service Host Subscriber Entry



If you select QXtend Web Service as the communication method, the Configuration Parameters window contains the following additional fields:

Tomcat Host. Specify the hostname of the Tomcat server that hosts the Web service.

Tomcat Port. Specify the port number where the Web service is available.

Webapp Name. Specify the name of the Web service used to connect to QXtend Inbound.

SSL. Select the field to use Secure Socket Protocol https encryption on messages to and from the Tomcat server.

Note If SSL is selected, then Tomcat Host must include domain name of the host.

3 Identify the profiles used to format QDocs for this subscriber by clicking on Register Profiles. A standard lookup displays, listing available profiles and those currently assigned. Use the Add and Remove buttons to move selected profiles from one list to the other.

4 If you want to restrict this subscriber to receiving data from one or more given source applications, click Register Src Apps. Add the source applications this subscriber should receive data from. When none are specified, the subscriber can receive from all valid source applications.

5 Click Save to save and exit the Configuration Parameters screen. The new subscriber displays in the overview screen.

Note Profiles assigned to a subscriber are listed under the subscriber in the left-hand navigation menu.

If the subscriber is new, you can click Force Publish in the Subscriber overview screen to force the publication of a QDoc for every record associated with a profile or set of profiles. Force Publish queries every record in the source application that fits the filter criteria of each selected profile and creates an event for them.

You can also click Generate Schema to generate the XML schemas for every profile registered to this subscriber.

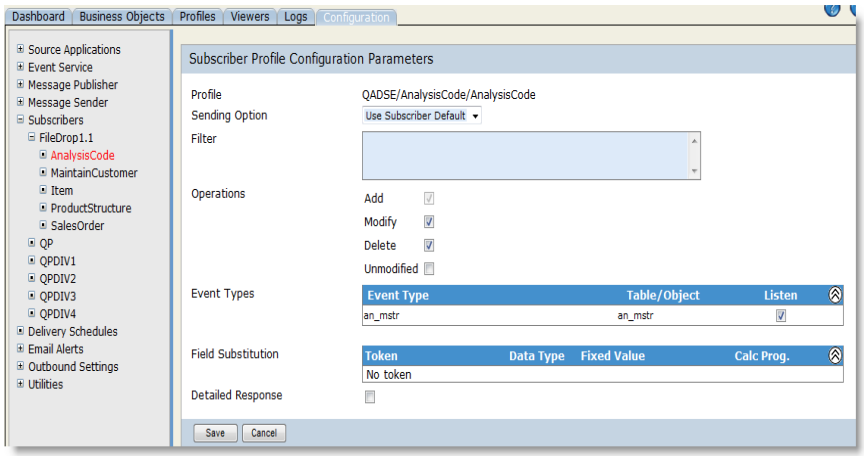
Configuring Subscriber Profiles

You now can configure individual profiles for subscribers to enhance your control over when messages are sent. For example, you can specify which events must occur before a message is published to a subscriber. The message will be sent only when the event types you select were the original trigger for the raw message.

You can define a sending option, and filter based on data within the profile message, and substitute values for tokens in the profile message. You can also define which operations—Add, Modify, and so on—the subscriber requires.

You configure subscriber profiles using the Subscriber Profile Configuration Parameters screen. To display this screen, click the name of the profile under the subscriber to which the profile is registered.

Fig. 2.31
Subscriber Profile Configuration Parameters



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 dropdown of the defined schedules.

Filter. Enter a valid Progress query to use with the subscriber profile. 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. On saving the subscriber profile, the system validates the query and returns an error message if the query expression is not valid.

Note If a subscriber profile is modified, every subscriber that registers that profile must validate the filter. If the filter results in an invalid query, a warning is displayed.

Operations. By default, all operations are selected except Unmodified. At least one option must be selected; otherwise the system displays an error.

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. If the filter query produces a Progress error, the system issues an alert. The operation is based only on the top-level table.

Event Types. Displays the event types associated with the subscriber profile; these are the active event types for all the data objects in the related business object. The Listen check box must be selected to activate an event type. By default all event types are selected.

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. To edit a token, select the check box to the left of the token name and click Edit. Then enter a fixed value and/or calculated program as required.

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. The tokens, which come from the profile, are signified by an “at” symbol (@) on either side of the variable, and are placed in the fixed value fields. The message publisher determines if any tokens need replacing in a profile message and creates a different profile message for each subscriber. For example, each subscriber may require a different fixed value “Site,” and the remainder of the profile data to remain the same.

Token. Specify any required tokens to set values on the profile based on the subscriber that is receiving them.

Data Type. Displays the data type of the field: character, date, datetime, datetime-tz, decimal, int64, integer, or logical.

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.

You can double-click in the Fixed Value field and view or enter data in a pop-up window.

The fixed value you provide must be consistent with the data type specified for the field. If there is a mismatch, the system displays an error message when you try to save the subscriber profile configuration parameters.

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.

Make sure the calculated value will be consistent with the data type specified for the field. If there is a mismatch, the system does not prevent you from saving the subscriber profile configuration parameters, but will run into errors during data publishing.

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

Yes: suppressResponseDetail is set to false.

No: suppressResponseDetail is set to true.

For detailed information and the effects of the suppressResponseDetail attribute, see “suppressResponseDetail” on page 288.

Subscriber Visibility

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.

Example Table 2.2 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.

Table 2.2
Subscriber Visibility Scenario

Description	Price	Subscriber 1: All orders	Subscriber 2: Over $5,000
Initial order	$4933	✓	
Add item	$5345	✓	✓
Reprice with 10% discount	$4810	✓	
Add on-site assembly	$6250	✓	✓
Split assembly to separate SO	$4810	✓	

Subscriber 1 sees all the changes, while Subscriber 2 only sees the two changes that temporarily raise the so_price value above $5,000.

QXO Delivery Schedules

A delivery schedule can be defined to control the transmission of a single document or multiple documents in a batch to subscribers at one or more specific times. You can select one delivery schedule to be used as the default schedule.

Outbound messages are sent only if they exist—that is, if the delivery time defined in a delivery schedule arrives and there are no outbound messages, nothing is sent.

Various types of schedule can be defined for transmitting documents:

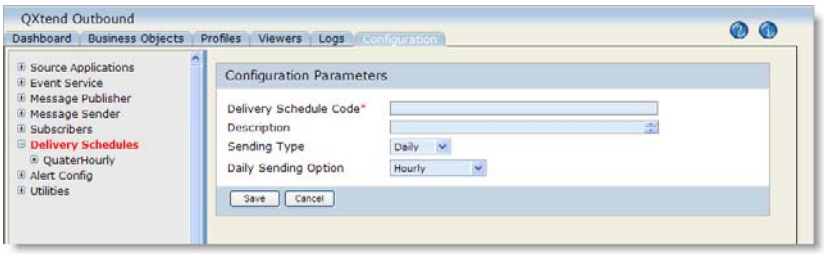
- Immediately when published. If no delivery schedules are defined this is the default system behavior. You select this option when defining a subscriber. See “QXO Subscribers” on page 38.

- At a specified time (or times) during the day. You can choose from several predefined times or specify one or more user-specified times.
- On selected days of the week at specified times.
- During selected months on specified days at specific times.

Example The QXO system administrator wants to configure a delivery schedule to send documents each hour during business days. A delivery schedule called BusinessHours is created and configured thus: the Sending Type is Weekly; the days Monday, Tuesday, Wednesday, Thursday, and Friday are selected; Sending Times are specified for 08:00 – 17:00.

1. In the Configuration tab, click Delivery Schedules in the navigation tree. The Delivery Schedules overview screen displays. Use this screen to select the delivery schedule the system should use by default.
2. Click New to create a new delivery schedule. Figure 2.32 shows the configuration options available for the Daily sending type.

Fig. 2.32
Delivery Schedule Configuration Parameters

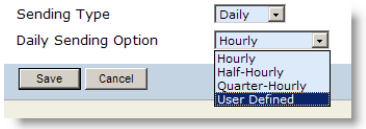


Delivery Schedule Code. Enter a code (up to 10 characters) that identifies the delivery schedule.

Description. Enter a description (up to 35 characters).

Sending Type. Choose a sending type for the outbound messages.

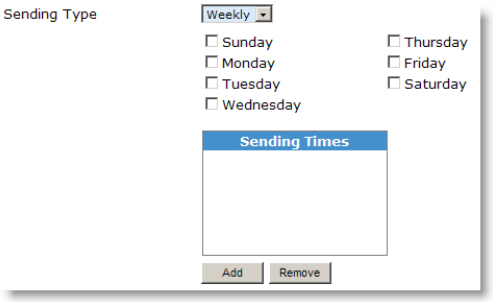
Fig. 2.33
Daily Sending Type



If Sending Type is Daily, outbound messages are sent daily.

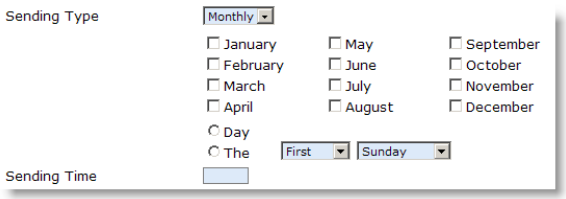
Daily Sending Option. Choose a daily sending option (Hourly, Half-Hourly, Quarter-Hourly, or User Defined). If User Defined is selected, specify a sending time or times by clicking the Add button. Sending times must be entered in 24-hour format. To remove a specified time, select the time and then click Remove.

Fig. 2.34
Weekly Sending Type



If Sending Type is Weekly, outbound messages are sent weekly. Select the days of the week on which to send the outbound messages. To add a sending time, click Add and then type a time into the New Time field. To remove an existing time, select the time you want to remove and then click Remove.

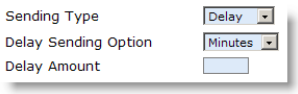
Fig. 2.35
Monthly Sending Type



If Sending Type is Monthly, outbound messages are sent monthly. Select the months in which to send the outbound messages. Select the Day option and enter a date to specify a specific day, or select the "The" option to specify a day; for example, the First Sunday of the month. Then enter a time in the Sending Time field.

Note If a value is entered in the Day field that is invalid for one or more selected months, outbound messages are sent on the last day of the month. For example, if 31 is entered in the Day field and the months January, February, and April are selected, the outbound messages are sent January 31, February 28 (during non-leap years), and April 30.

Fig. 2.36
Delay Sending Type



If Sending Type is Delay, the system postpones sending subscriber messages by a specified amount of time after message creation.

Delay Sending Option/Delay Amount. Specify the amount of time after message creation to publish subscriber messages.

3 Click Save to save the delivery schedule configuration.

QXO E-mail Alerts

The e-mail alerts feature allows specified recipients to receive e-mail alerts that are raised by QXO for events relating to specific subscribers and profiles.

Alerts in QXO are generated by three types of events:

- Default (system) alerts, which are triggered by QXO at a stage in processing that does not yet involve a specific subscriber or profile. Default alerts may arise during event extraction, during the creation of business objects by the event service, when services are started with no subscribers configured, or when the delivery of a processing alert fails.
- Processing alerts, which are triggered by processes involved in the handling of events or during QDoc creation. Processing alerts may arise if, for example, a required business object is not registered with a message publisher, or if the creation of a subscriber profile message fails.
- Metric-related alerts, which are triggered when MSG thresholds for metrics are exceeded for an event, message publisher, or message sender service. For details of these metrics see “QXO Services” on page 32.

Default alerts and metric-related alerts are typically used by system administrators to facilitate QXO configuration troubleshooting.

E-mail alerts are delivered using simple mail transfer protocol (SMTP).

Note Similar e-mail alert functionality—tailored to the processing of inbound documents—is available in QXI. For details see “QXI E-mail Alerts” on page 192.

In the QXO Console you can:

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

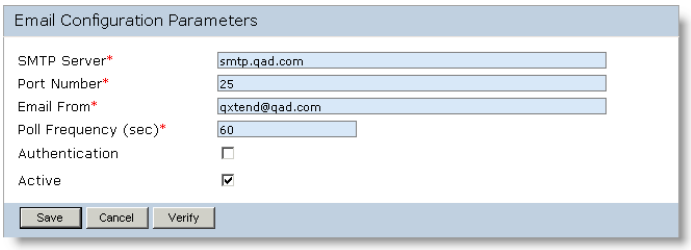
Configuring E-mail Settings

Use the Email Configuration Parameters screen to configure e-mail settings. You can validate your e-mail setup by clicking the Verify button. The process for configuring e-mail settings in QXO is the same as that in QXI.

Note Before configuring e-mail settings, ensure that access is available to the SMTP server.

1. In the QXO Console click the Configuration tab.
2. Click the Email Alerts node in the left-hand tree view. If it was not already displayed, the Email Configuration Parameters screen displays.

Fig. 2.37
Email Configuration Parameters



3 Enter the following values as required.

SMTP Server. Specify the address of the SMTP server used to deliver the alerts.

Port Number. Specify the port number used to access the SMTP server.

Email From. Specify the address to use in the “From:” in the header of the e-mail message. This field is required.

Poll Frequency. Specify the pause time before next check when the service does not detect any e-mail to be sent.

Authentication. Select this option if the SMTP server to be used requires authentication. When this check box is selected, the User Name and Password fields display.

User Name. If authentication is required, specify a user name.

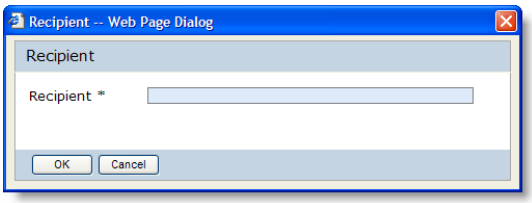
Password. If authentication is required, specify a user password.

Active. Select this option to activate e-mail alerts and to create the e-mail service. No e-mail alerts will be distributed unless this option is selected.

Note If this option is not selected alerts will still be stored in `qxodb`.

Verify. Click to validate the SMTP configuration settings. The Recipient dialog displays.

Fig. 2.38
Recipient



Enter an e-mail address in the Recipient field and click OK. If the SMTP server is available and can be communicated with, a success message is sent to the specified address and a message appears on the Recipient dialog. If the setup is incorrect—or if the SMTP server cannot be communicated with—an error message displays.

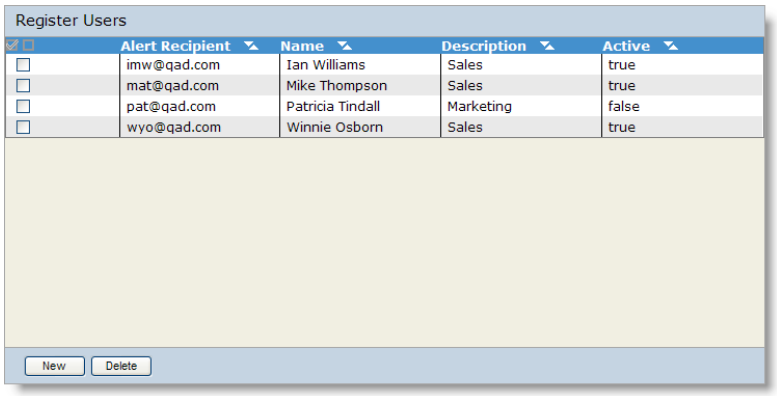
Managing Alert Recipients

Use the Register Users screen to view the e-mail recipients currently defined in QXO. You also can add or delete recipients using this screen.

1 In the QXO Console, click the Configuration tab.

2 Choose Email Alerts|Recipients to view the currently defined recipients.

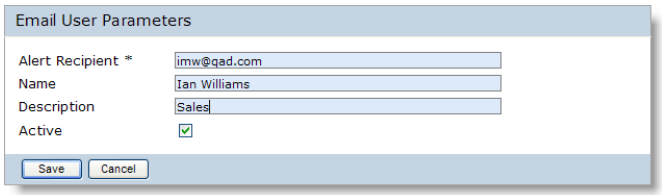
Fig. 2.39
Register Users



Add an Alert Recipient

1 Click New on the Register Users screen to display the Email User Parameters screen.

Fig. 2.40
E-mail User Parameters



2 Enter the following values as required.

Alert Recipient. Specify the e-mail address of the alert recipient. This field is required.

Name. Specify the name of the alert recipient.

Description. Enter a brief description as needed.

Active. Select this option to activate the recipient. The recipient will not receive e-mail alerts unless this option is selected.

3 Click Save to save the new recipient.

Managing Alert Groups

Use the Register Alert Groups screen to view the alert groups currently defined in QXO and the number of members in each group. You also can add or delete alert groups using this screen, as well as define the message configuration (template) if required.

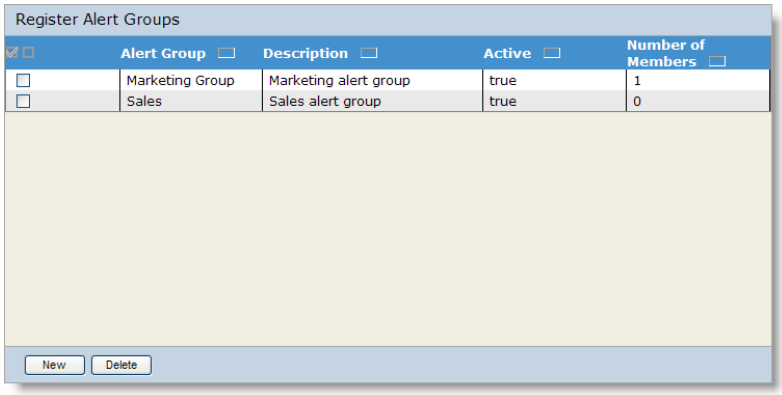
Managing an alert group consists of:

- Adding an alert group
- Registering recipients
- Setting the alert options
- Registering subscribers and profiles

1 In the QXO Console click the Configuration tab.

2 Choose Email Alerts|Groups to view the currently defined alert groups.

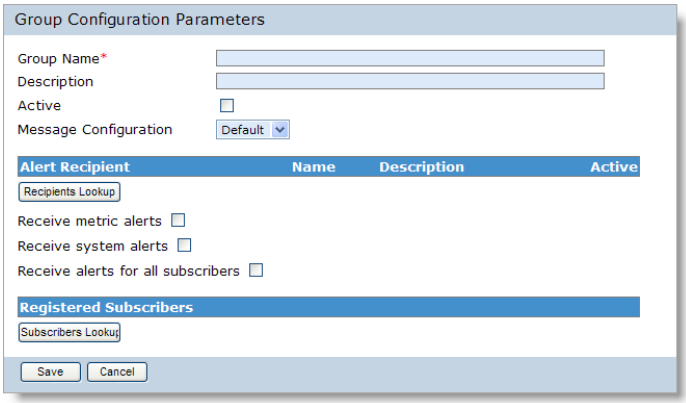
Fig. 2.41
Register Alert Groups



Add an Alert Group

1 Click New on the Register Alert Groups screen. The Group Configuration Parameters screen displays.

Fig. 2.42
Group Configuration Parameters



2 Enter the following values as required.

Alert Group. Specify a name for the alert group. This field is required.

Description. Enter a brief description as needed.

Active. Select this option to activate the alert group. No e-mail alerts will be distributed for this group unless this option is selected.

Message Configuration. Specify whether to use the default message configuration or a custom message configuration.

Default. The message will use the description stored in `qxodb`.

Custom. The message will use the subject and body you provide. Selecting this option causes the Email Subject, Email Body, and Attachments fields to display.

E-mail Subject. Enter a brief subject description as needed (for custom message configuration only); for example:

```
A QXtend alert has been triggered.
```

E-mail Body. Enter the text to use to precede the error message and attachments (for custom message configuration only). For example:

```
There has been a raised alert from QXtend. This event was triggered
automatically and the following message has been recorded:
```

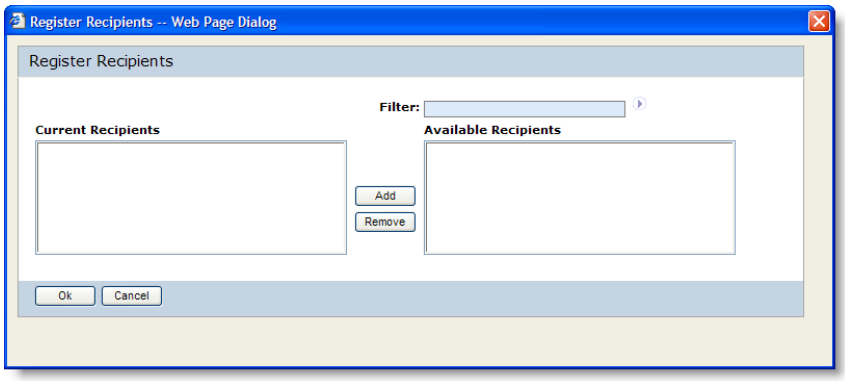
Attachments. Select the Request and/or Response check boxes to include the request and/or response as an attachment in the body of the e-mail alert (for custom message configuration only).

Register Recipients

1 Click Recipients Lookup to register the alert recipients to the alert group. The Register Recipients lookup displays. Click the arrow next to Filter to populate the Available Recipients list. Use the Add and Remove buttons to update the lists.

Note For details about using the Filter feature see “Filtering Names” on page 20.

Fig. 2.43
Register Recipients



2 Click OK to save and exit the Register Recipients lookup.

Select Alert Options

1 Select the Receive metric alerts option to generate e-mail alerts when defined MSG alert metrics have been exceeded for a service. For details on services see “QXO Services” on page 32.

2 Select the Receive system alerts option to generate e-mail alerts that are triggered by system-related processes.

Register Subscribers

1 Select the Receive alerts for all subscribers option to cause e-mail alerts to be generated for all subscribers. If you select this option, specify the severity of the exceptions—ERR, WRN, or MES—that will generate an e-mail alert. Click Save to save your alert group configuration.

Otherwise leave this option clear to specify which subscribers to register in step 2.

2 Click Subscribers Lookup to register subscribers to the group. The Register Subscribers lookup displays. Click the arrow next to Filter to populate the Available Subscribers list. Use the Add and Remove buttons to update the lists.

3 Click OK to save and exit the Register Subscribers lookup.

4 For each subscriber, indicate the severity of the exceptions that will generate an e-mail alert.

5 Click Save to refresh the left-hand tree menu.

Register Profiles

1 In the left-hand tree menu, click a group/subscriber. The Configuration Settings screen displays for the group/subscriber.

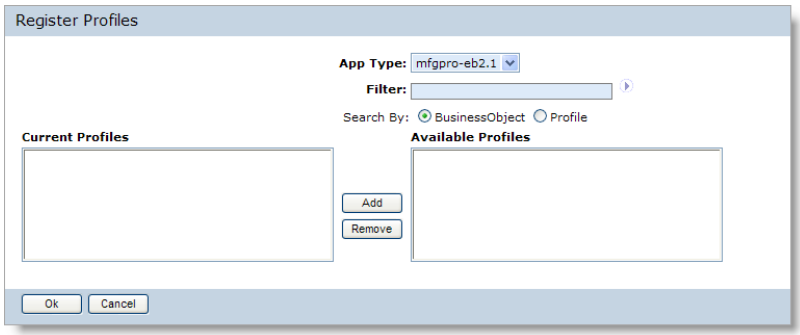
Fig. 2.44
Group/Subscriber Configuration Settings

2 Select the Receive alerts for all profiles option to cause e-mail alerts to be generated for all profiles. If you select this option, specify the severity of the exceptions that will generate an e-mail alert. Click Save to save your alert group configuration.

Otherwise leave this option clear to specify which profiles to register in step 3.

3 Click Profiles Lookup to register the profiles to the group/subscriber. The Register Profiles lookup displays. Select the application type and then click the arrow next to Filter to populate the Available Profiles list. Use the Add and Remove buttons to update the lists.

Fig. 2.45
Register Profiles



4 Click Save to save and exit the Register Profiles lookup.

5 Indicate the severity of the exceptions that will generate an e-mail alert.

6 Click Save to save the alert group.

Email Service

Use the Services|Email Service option on the Dashboard to view information about the existing e-mail service, such as e-mails sent, alerts pending, and so on. Use the Start and Stop buttons to control the e-mail service as required.

Fig. 2.46
Email Service

Note If you stop an e-mail service, the e-mail service stops during the next poll, according to the configured poll frequency. This can result in a delay between issuing the Stop command and the service registering the stop request. The longest possible delay is equal to the configured poll frequency.

For details about services, see “Viewing Specific Services” on page 99.

Importing Business Objects and Profiles

Using the XML Import feature under the Utilities node in the Configuration tab, you can import XML documents. This is a required step 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. See “Managing Business Object XML Files” on page 85 for information on dumping XML from QXO.

Prior to running this load, the source applications must be set up in QXO using the correct naming convention.

Important The predefined source application types are used to load the QAD standard business objects and profiles. These source application types are named:

- mfgpro-eb
- mfgpro-eb2
- mfgpro-eb2.1
- QADSE
- QADEE

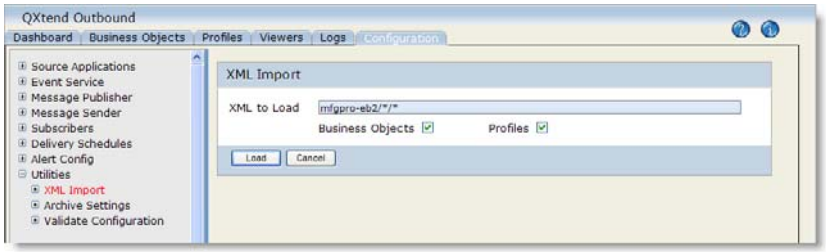
The standard QAD objects to be loaded reside in:

QXOsrvInstallDir\boXML

This is also the directory location used when you export business objects and profiles as described in “Managing Business Object XML Files” on page 85. You must copy the files from the originating QXO server to the target server prior to loading them into another instance of QXO.

1 In the XML Import screen under the Configuration tab, enter the source application type as defined in Source Applications, and two asterisks to denote a full import for both business objects and profiles. For example, for eB2 enter:

Fig. 2.47
XML Import of QAD Business Objects and Profiles



2 Select both checkboxes and choose Load. A processing note displays during the load.

3 After the load, messages display to inform you of the successful loads of each object.

To load specific files, such as those originating from another instance of QXO, follow the same steps, but enter the source application and the specific object names you want to import.

Fig. 2.48
XML Import of Custom or Individual Objects

Data Archive

Using the Archive Settings feature under the Utilities node in the Configuration tab, you can configure archive settings. You can delete and optionally archive data in the `qxodb` database that is no longer necessary for the day-to-day running of QXO.

Using the archive feature you can:

- Archive raw data from business objects
- Archive profile messages, including child data such as responses and exceptions
- Archive alerts and logs
- Schedule archive runs
- Run an archive service

Your business requirements will dictate the types of data—alerts and logs, published messages, event messages—you need to archive and how often you should archive.

Archive Guidelines

Archiving allows you to constrain the increase in size of the `qxodb` database to acceptable limits while giving you access to message data should you require it. Typically organizations delete message data once it has been sent, but certain regulatory or legal requirements might necessitate message data to be archived for a certain period of time.

All data can be safely archived. QXO will retain the necessary data to ensure correct operations, and delta fields are created for subscriber messages.

Logs and alerts typically occupy minimal space in the database. However, your business requirements may require logs to be archived for auditing purposes.

Event messages occupy most space in the database as they often have associated child data. Event messages can be archived to provide a record of business object modifications.

Typically you would delete transaction messages as the transactional data will never be modified.

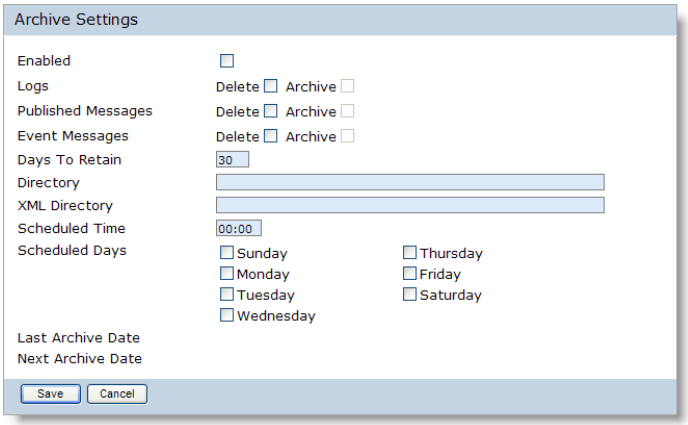
Archiving Using a Cron Job

You can also run the archive feature by using a cron job. In the crontab file, specify the archive program qxoarchive.p. Using a cron job lets you schedule archiving to occur at multiple times per day, if required. Configuring archiving using the Archive Settings screen constrains archiving to occur only once per day on the days you specify.

Note Use the archive service to start or stop the archive routine and to view its current status. You do not have to create an archive service—you just have to enable it. For details on using services, see “Viewing Services Summaries” on page 99.

1 In the Configuration tab, click Utilities|Archive Settings in the navigation tree. The Archive Settings overview screen displays.

Fig. 2.49
Archive Settings Configuration



Enabled. Indicates whether the archive routine is enabled. To run the archiving routine this check box must be selected, regardless of whether archiving is configured from the Archive Settings screen or scheduled using a cron job.

Logs. Indicates whether archiving is enabled for logs.

Delete. Indicates whether to delete the archived logs. You must select this check box in order to enable the Archive option.

Archive. This option is enabled when the Delete check box is selected. Indicates whether to archive logs to a file.

Published Messages. Indicates whether archiving is enabled for published messages and associated child data.

Delete. Indicates whether to delete the archived published messages. You must select this check box in order to enable the Archive option.

Archive. This option is enabled when the Delete check box is selected. Indicates whether to archive published messages to a file.

Event Messages. Indicates whether archiving is enabled for event messages and associated child data.

Delete. Indicates whether to delete the archived event messages. You must select this check box in order to enable the Archive option.

Archive. This option is enabled when the Delete check box is selected. Indicates whether to archive event messages to a file.

Days To Retain. Indicates the number of days the data will remain in the QXO database before it is archived. The default value is 30; the maximum value permitted is 999.

Directory. Specify a location for the archive files. This directory contains all archived data except subscriber messages and responses (child data). The directory must exist and be writable before the archive routine is started. If not, the archive routine fails and the failure is logged.

The archive routine outputs the data into a file called <*table-name*>.<*yyyymmdd*>.arc where <*table-name*> is the name of the application table being archived and <*yyyymmdd*> is the creation date. If the archive routine is run more than once in a day, any archived data is appended to the end of the existing archive file.

XML Directory. Specify a location for the archived subscriber messages and responses. This directory is populated only if you specify that you want to archive published messages. This directory must exist and be writable. If this field is left blank, by default any data is saved to the location specified in the Directory field.

Scheduled Time. Enter a time to commence the archive routine. Time must be entered in a 24-hour format.

Scheduled Days. Select the days on which to run the archive routine.

Last Archive Date. Displays the date on which the archive routine was last run.

Next Archive Date. Displays the date on which the archive routine is next scheduled to run based on the Archive Settings configuration.

Validating QXO Configuration

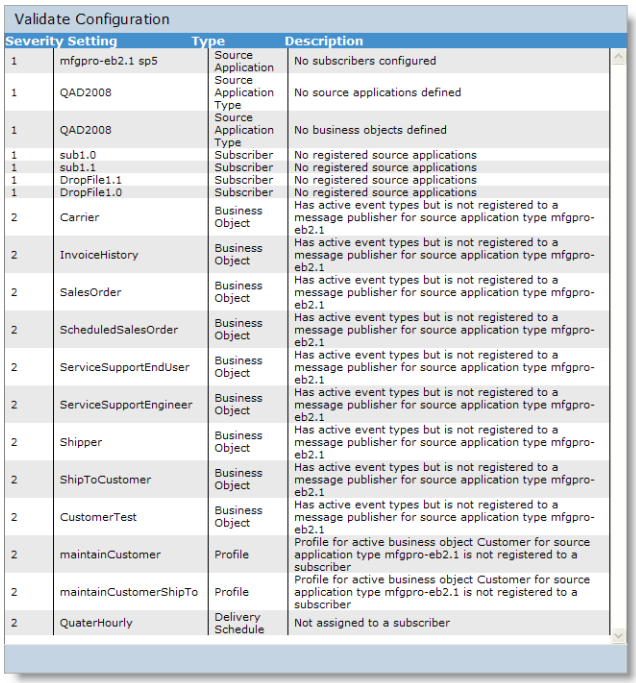
Using the Validate Configuration feature under the Utilities node in the Configuration tab, you can validate your QXO configuration. Validating your configuration helps you identify and rectify any issues with your QXO setup.

Using the validation feature you can validate your configuration for the following:

- Source applications
- Business objects
- Profiles
- Subscribers
- Delivery schedules
- Archive settings
- Event types

1 In the Configuration tab, click Utilities|Validate Configuration in the navigation tree. The Validate Configuration screen displays.

Fig. 2.50
Validate Configuration



The Validate Configuration utility provides validation information—severity, message, and cause—about each aspect of your QXO configuration. A severity level of 1 indicates a critical error. A severity level of 2 is an advisory to check your configuration.

Table 2.3 shows the types of information provided.

Table 2.3
Configuration Messages

Severity	Message	Cause
SOURCE APPLICATION TYPES		
1	No source application types defined	No source application types have been set up in the Configuration/Source Applications menu.

Severity	Message	Cause
1	No source applications defined	No source applications have been set up for a source application type in the Configuration/Source Applications menu.
1	No business objects defined	The default business objects have not been loaded for the source application type, and none have been manually entered.
SOURCE APPLICATIONS		
1	No databases defined	No databases have been set up in the configuration/source applications/databases menu.
1	Could not connect to the qxevents database	Either the qxevents database configuration is incorrect or the database is not running.
2	No domains defined	In a source application with a domain enabled QAD Enterprise Applications database, no domains have been set up in the Configuration/Source Applications/Domains menu.
1	No event types defined	The default data has not been loaded into the qxevents database.
1	No event types are active	No event types in the Configuration/Source Applications/Event Types list have been checked as activated.
1	No event service configured	There are either no event services set up, or no existing event service has the source application registered.
1	No subscribers configured	There are either no subscribers set up, or no existing subscriber has the source application registered.
SOURCE APPLICATION DATABASES		
1	<*Progress database connection message*>	Error connecting to the database. Either it is incorrectly configured in the settings, or the database is not running.
DOMAIN		
1	Does not exist in QAD Enterprise Applications for source application <*src-app*>	The source application has a domain configured, but it does not exist in the QAD Enterprise Applications database for this source application.
EVENT TYPES		
2	Has no related business object for source application <src-app>	An event type has been set to be activated, but there is no business object that uses it.
2	Not registered with event service for source application QADERP	The active event type is not registered with any event service.
BUSINESS OBJECTS		

Severity	Message	Cause
2	Has active event types but is not registered to a message publisher for source application type <*src-app-type*>	A business object has an activated event type but has no message publisher, so it will not be published.
1	Registered to message publisher <*msg-publisher*> but has no active event types	A business object is registered to a message publisher but there are no active event types, either because they are not activated or the data watch is turned off.
BUSINESS OBJECT Group		
2	Related event types are not registered with the same event service.	If active event types in a business object group are not registered with the same event service, business objects in the group will be processed by different event services, which defeats the purpose of business object groups since you cannot guarantee that data of designated business objects will be processed as set up in the group in a single-thread manner.
PROFILES		
2	Profile for active business object <*bo-name*> is not registered to a subscriber	The business object is active, and a profile other than the default has been created, but it has not been registered with a subscriber.
1	Registered to subscriber <*sub-name*> for business object <*bo-name*> but has no active event types	The business object this profile is related to is not active yet the profile is registered to a subscriber.
SUBSCRIBERS		
1	No registered profiles	The subscriber has no registered profiles.
1	No registered source applications	The subscriber has no registered source applications.
1	No message sender defined	The subscriber is not registered with any message senders.
DELIVERY SCHEDULES		
2	Not assigned to a subscriber	The delivery schedule is configured but is not being used by any subscribers.
ARCHIVE		
2	Configured to delete event messages but no active business objects are set to archive	The archive service will delete event messages if directed, however each individual business object has an archive flag, and in this case no active business objects have that flag set.

Implementing Calculated Fields

A calculated field holds a value that is the result of a mathematical operation performed by a program. The calculated field program can be placed in the QXO service PROPATH—that is, defined in `start-sess.sh`—or run from an AppServer if the calculation requires input from a database. If it is a simple calculation that does not require access to a database, place it in the QXO service PROPATH, as this is faster. If it requires database access, use the AppServer.

If connecting to a database, calculated fields require correct connection parameters for the AppServer. See “QXO Source Applications” on page 23 for details on setting up AppServer parameters.

You can find sample calculated field programs in `<QXOSrvInstallDir>/src/samples`. One of those sample programs, `samplecalc1.p`, is listed here.

```
define input parameter dataset-handle phDataset.
define input parameter pcBuffer as character.
define input parameter pcQueryString as character.
define output parameter pcResult as character.
pcResult = string(now).
delete object phDataset no-error.
```

This program is provided as an example of the format that a calculated field program should take. The important part is the signature and the input and output parameters:

- The first parameter is the entire profile message defined as a dataset.
- The second parameter is the name of the buffer in the dataset that is currently in scope.
- The third parameter is the query string for that buffer that locates the exact record in the dataset that is in scope.
- The fourth parameter is the result of the calculation.

The AppServer Parameter field on the source application configuration contains all the necessary parameters for Progress to connect to the AppServer. A typical set of AppServer parameters might be:

```
-AppService qxoappserver -H co9906
```

This allows QAD QXtend to access the calculated field program on an AppServer named `qxoappserver` on the machine `co9906`. The contents of this field are executed in the code like the following:

```
create server hAppServer.
hAppServer:connect("-AppService qxoappserver -H co9906").
run value("myprogram.p") on hAppserver (input dataset-handle phProfile by-value, input
pcBuffer, input pcQueryString, output output pcResult).
```

If you are implementing a calculated field program that accesses the QAD Enterprise Applications databases, the program should use an application server separate from the one defined for QXO. For simple calculated field programs where the logic applied is local to the calculation program, it is safer and faster to locate the program in the normal code base.

Calculated Fields Program Example

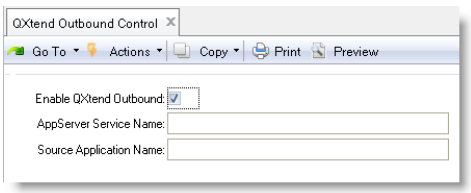
The following program used by QXO is an example of a calculated field program. The code converts the value stored in a database field into another format in the outgoing QDoc. The resulting format is suitable for use in an incoming QDoc through QXI.

```
$RCSfile: cp-pipartcode.p,v $
Copyright 1986-2004 QAD Inc., Carpinteria, CA, USA.
All rights reserved worldwide. This is an unpublished work.
Description: Checks the value of the pi_part_code field extracted from QAD Enterprise
Applications. If it matches qadall variable we require to place blank for pi_part_code in
the published business object

============================================================

define input  parameter dataset-handle phDataset.
define input  parameter pcBuffer      as character.
define input  parameter pcQueryString as character.
define output parameter pcResult      as character.

define variable hQuery  as handle no-undo.
define variable hBuffer as handle no-undo.
define variable hField  as handle no-undo.

define variable cQadAll as character initial "qadall--+--+--+--+--+" no-undo.
define variable cValue  as character initial "" no-undo.

hBuffer = phDataset:get-buffer-handle(pcBuffer).

create query hQuery.
hQuery:add-buffer(hBuffer).
hQuery:query-prepare(pcQueryString).
hQuery:query-open().
hQuery:get-first().

if hBuffer:available then do:
    hField = hBuffer:buffer-field("piPartCode") no-error.
    if valid-handle(hField) then
        cValue = hField:buffer-value.

    if cValue = cQadAll then
        cValue = "".

end.

hQuery:query-close().

pcResult = cValue.
delete object phDataset no-error.
```

Setting Up Outbound Control in QAD Enterprise Applications

Use QXtend Outbound Control (36.16.19) in QAD Enterprise Applications to configure QXO settings.

Fig. 2.51
QXtend Outbound Control (36.16.19)



Enable QXtend Outbound. Specify whether any changes in QAD Enterprise Applications will trigger events to QXtend.

AppServer Service Name. Specify the AppServer service name for Direct Data Publish when you want to create a program to call the DDP API provided in QAD Enterprise Applications to publish data to QXtend. To use this feature, please contact QAD Services.

Source Application Name. Specify the source application name for Direct Data Publish when you want to create a program to call the DDP API provided in QAD Enterprise Applications to publish data to QXtend. To use this feature, please contact QAD Services.

Chapter 3

QXtend Outbound Business Objects

The following material covers implementation of QAD QXtend Outbound (QXO) business objects.

Introduction

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, even though 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.

You define a business object in QXO that encompasses all the possible attributes of a particular type of document, order, or data set. You define which source application tables you want, how the tables are related to each other, and which fields in each table to include. You can define custom fields as well, to contain either a fixed value every time the business object is updated, or a value calculated at each update.

You can define business objects as DDP, regardless of whether or not the source application is DDP. Tables and fields in DDP business objects cannot be added, modified, or deleted—the business object itself can be deleted. For details see “Modifying Business Objects” on page 73.

Not all subscribers for your system need the entire business object. Most subscribers need distinct components of a business object; shipping needs address and inventory data, sales needs sales figures, accounting needs sales and tax data, and so on.

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 original business object remain unchanged.

QAD and Custom Business Objects and Profiles

QXO is shipped with a selection of critical business objects and profiles. These can either be used as they are—or more likely—copied and used as templates for your own business objects and profiles.

For a listing of standard QXO business objects, see Table 1.1 on page 5.

Important Only an experienced professional with an understanding of the target application database structure should customize QAD-supplied business objects. QAD does not support custom or non-QAD-defined business objects. To ensure support for your customizations, engage QAD Services personnel to create new or modified business objects.

Typically, a business object has one primary table and other related tables that are joined to it. The business object definition identifies the join relationships. You can also specify filters to limit the data retrieved from the joined table. Then for each field in the table you can specify:

- Whether the field is always published regardless of profile settings
- Whether the field is part of a primary table index

- Any fixed constant field values
- Programs to call to return the value for the field

When you start with a copy of a QAD business object or build your own, you can add additional tables and fields—as long as that business object does not employ DDP. You can also add your own custom fields to tables. These represent static values that you want to include whenever data from this business object is published. If, for example, you are designing a business object that will create an outbound QDoc that corresponds to an inbound QDoc, you might need to include values for fields that control the behavior of the QAD Enterprise Applications UI and that are not stored in the database.

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.

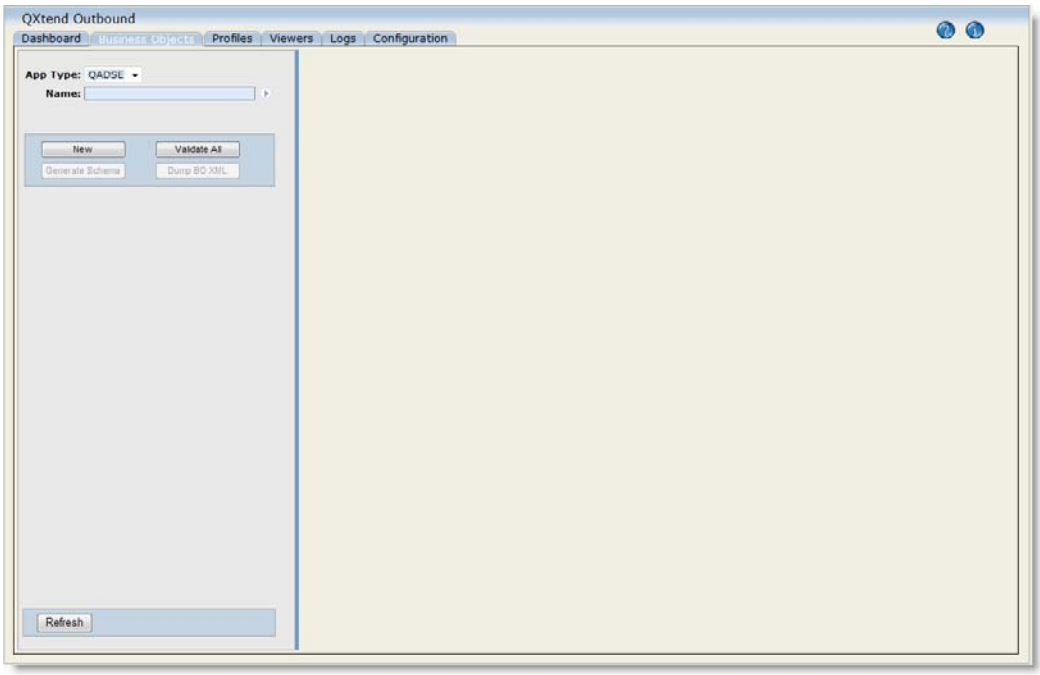
Note You cannot copy and paste data objects from business objects that employ DDP.

Profiles are views of business objects tailored for the requirements of specific subscribers. You use profiles to select which components of a business object are sent to subscribers. The method for defining a profile is nearly identical to the method for defining a business object; however, unlike business objects, profiles always start as a copy of a default profile.

Viewing Business Objects

When you click the Business Objects tab, the screen illustrated in Figure 3.1 displays.

Fig. 3.1
Business Objects

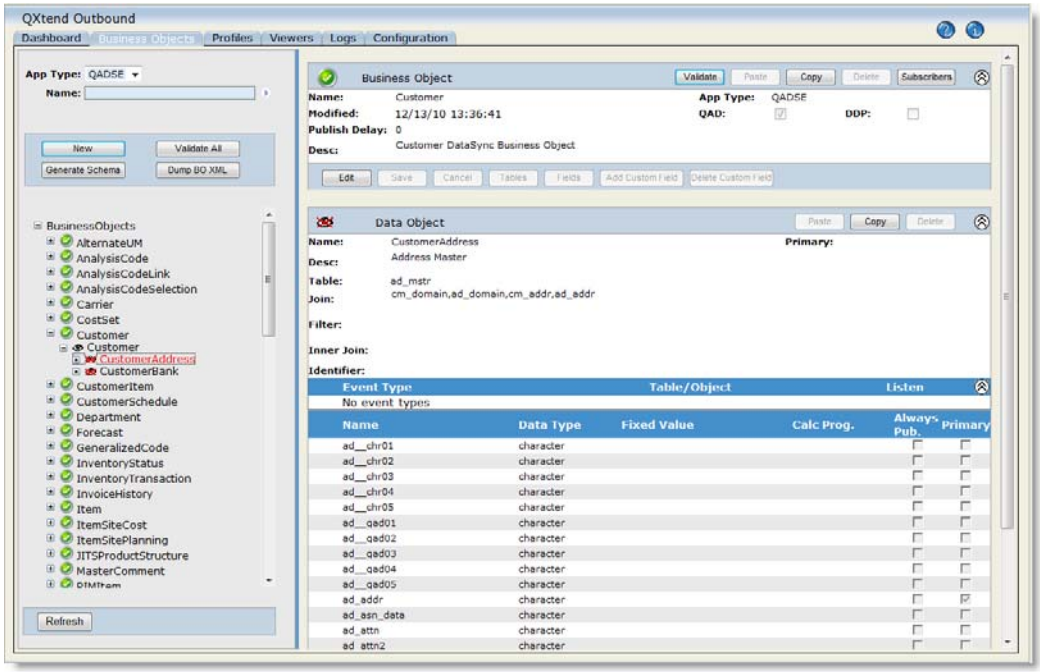


App Type. Select the source application type from the drop-down list. Source applications are defined in the Configuration tab. The last selected value is saved and used as the default entry the next time this screen is viewed. See “QXO Source Applications” on page 23.

Name. Use the Name field to find business objects to view or update. See “Filtering Names” on page 20.

Click the arrow next to Name to display all available business objects with the Name field left blank, or to search within the available business objects for objects that match your Name entry. Objects display in a tree view as shown in Figure 3.2.

Fig. 3.2
Business Object Tree View



Performance Considerations

Each business object contains one or more data objects maintained in a hierarchy. The entire set of data objects under a business object is referred to as a *business object tree*. Since QXO must search up and down the tree for different functions—at least for business objects that do not employ DDP—each tree is validated in both directions during the initial load of business objects.

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.

When you are populating a business object tree with a data set—for example, during a forced load of data from your source application to one or more subscribers—QXO knows the business objects required, but not all the data objects. In this case, it searches down the tree. Where the indexes are inadequate, a whole-index search occurs. However, these are generally one-time events and are not a performance concern.

When an event occurs in a source application, QXO knows which data objects are impacted, but not which business objects they belong to. In this case, the search goes up the tree. These searches occur during production processing and are of greater concern. Where the indexes are inadequate, a reverse whole-index search occurs.

Note In the case of event data for business objects that use DDP, the raw business object is published to QXO and performing a search is not required, significantly reducing processing time.

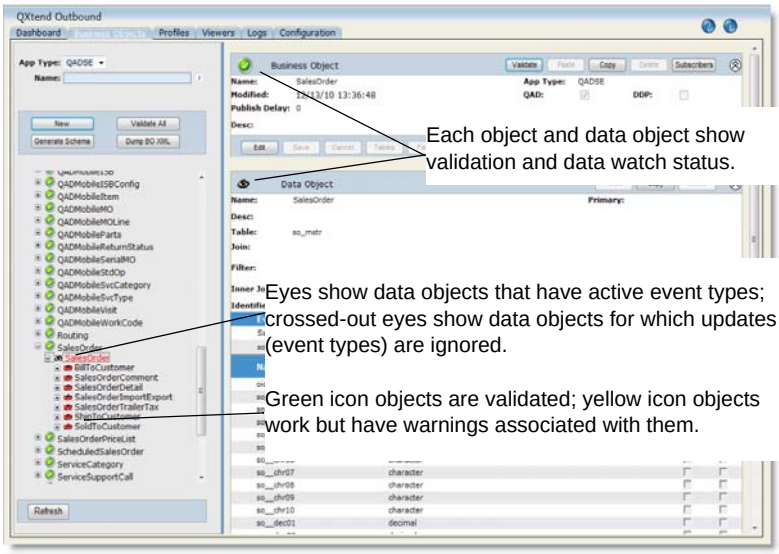
Business Object Validations and Data Watches

Both types of searches are tested during business object validation. A validation occurs against the connected source applications during the initial load of business objects—at least for business objects that do not employ DDP. The result of this validation determines the color of the icon of each business object in the business object navigation tree.

- A green icon shows that the validation was successful.
- A yellow icon displays when reverse whole-index searches are required for one or more data objects in the business object tree during one or both of the validation passes.
- A red icon displays when invalid joins appear, or when a business object has not been validated.

To control which data objects are watched—that is, the data objects that have active event types that are being “listened” to—an Event Type section displays on each data object. Select the Listen check box to determine which event types you want to be active on the data object. Event types essentially operate as multiple data watches on a data object. QAD has set this option for the default business objects to minimize the number of reverse whole-index searches and yet retrieve all essential business data. Two business objects—Customer Schedules and Sales Orders—still have unavoidable reverse whole-index searches.

Fig. 3.3
Validations and Data Watch Symbols on Business Objects



When you open a business object node in the navigation tree, each data object has an eye icon.

- An eye alone means that the data object is watched; that is, the object has one or more active event types. A watched data object is aware of updates to that object's tables in the source application.
- An eye crossed out means that data object is unwatched; that is, either the data object has no defined event types, or its event types are all inactive. An unwatched data object ignores updates to the object tables, but the data for that object is still retrieved when events affect other data objects in the parent business object.

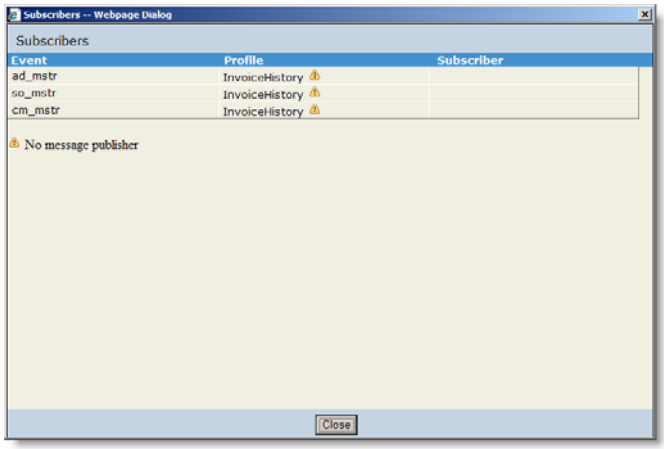
To modify these or other business objects, see the section "Modifying Business Objects" on page 73.

Viewing Business Object Subscribers

In order to determine more easily which subscribers will receive messages based on a business object—and the events that will trigger messages—you can display a Subscribers report. The Subscribers report lists the following:

- Activated event types
- Profiles related to this business object that also have this event type enabled
- Subscribers that receive these profiles based on the event type

Fig. 3.4
Subscribers Report



To display the Subscribers report, display the Business Objects tab, and then click the Subscribers button at the top-right corner of the business object configuration screen.

If a profile is not assigned to a message publisher, a warning symbol displays next to it. Similarly, if a subscriber is not assigned to a message sender, a warning symbol displays next to it. A tooltip for the warning symbols indicates the exact reason.

A warning icon is displayed next to any event type name that is not registered with any event service.

Modifying Business Objects

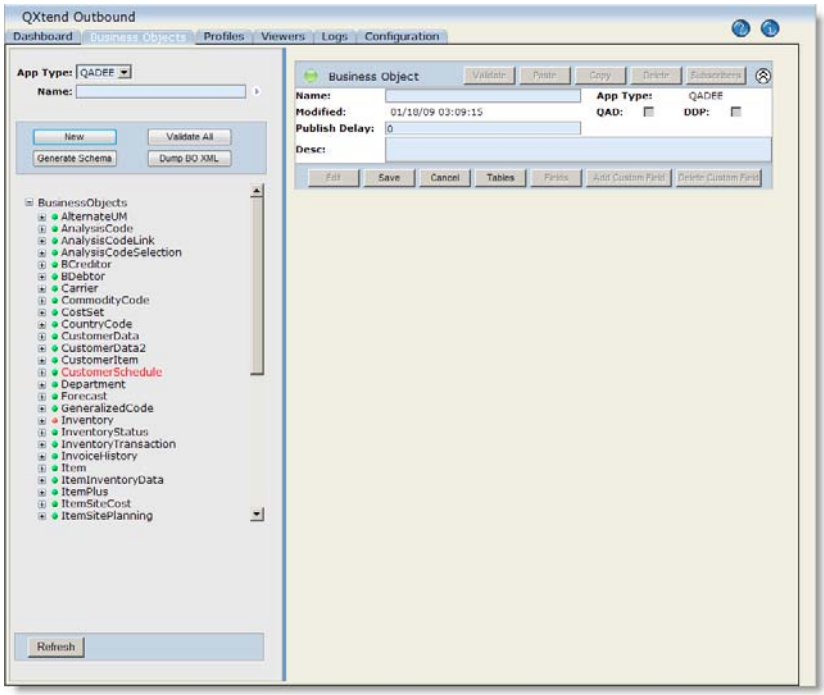
Since you cannot edit the QAD-supplied business objects, you must either copy an existing business object and modify it, or create a new one. When you create or copy a business object, a default profile of the same name is also created.

Note For business objects that employ DDP, only certain attributes can be modified. Those attributes that can be modified are indicated on the interface during edit mode.

1 Select the source business object from the tree view. If it is a QAD-supplied object, the only button active is Copy.

2 Click Copy. The new object is created including all related data objects. A new name is provided for the object by default.

Fig. 3.5
Business Object Definition Showing the Green (Validated) and Yellow (Warning) Dots



3 Enter values for the new business object as follows:

Name. Update the name as needed.

Modified. No edit allowed. This displays the date and time this business object was last saved.

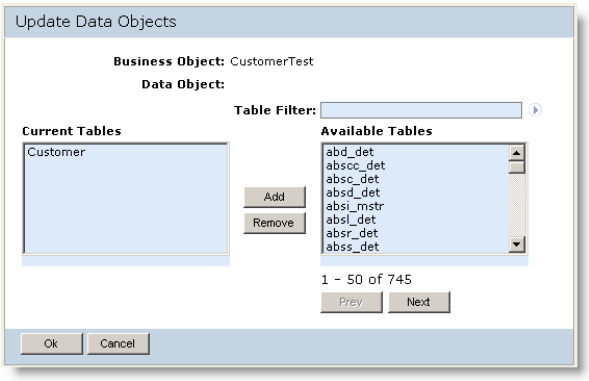
Publish Delay. Set this to a nonzero value for business objects that may generate multiple notifications to the `qxevents` database prior to completion of data entry. For example, sales order entry creates one or more notifications as different tables are updated before data entry is complete. Setting the value to 60 seconds or more allows time for data entry to complete and only the final data to be exported.

Desc. Enter a description of the object. Consider including the creation date, object author, and purpose.

This attribute may be modified for business objects that employ DDP.

4 To add or delete tables from the copied business object, click Tables. Update Data Objects displays. Click the arrow next to Table Filter to populate the Available Tables list.

Fig. 3.6
Business Object Table Selection



5 Select tables in either list and use the Add or Remove buttons to move the tables between lists. When the Current Tables list contains the tables you require for the business object, click OK. You return to the Business Object screen.

Note If you need to add new tables that are not listed, see “Creating New Business Objects” on page 76.

Prior to using the new business object, you must validate it and optionally modify the data watch attributes—that is, the active event types—on the data objects to improve performance.

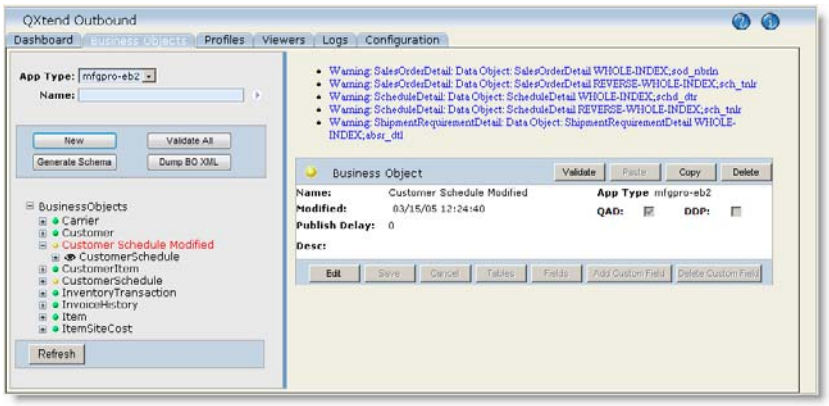
Validating and Optimizing Business Objects

When you copy or create a business object, the new object has a red dot next to it in the navigation tree. This means the business object has not yet been validated; it cannot be used until it has been validated.

Note Business objects that employ DDP are not validated.

1 On any new or copied business object, click Validate. Any validation warnings or errors appear above the business object definition. The whole-index warnings are WHOLE-INDEX for top-down searches where the business object is known but not the data objects, and REVERSE WHOLE-INDEX where the data objects are known but not the business objects they reside in.

Fig. 3.7
Validation Messages for a Copied Business Object

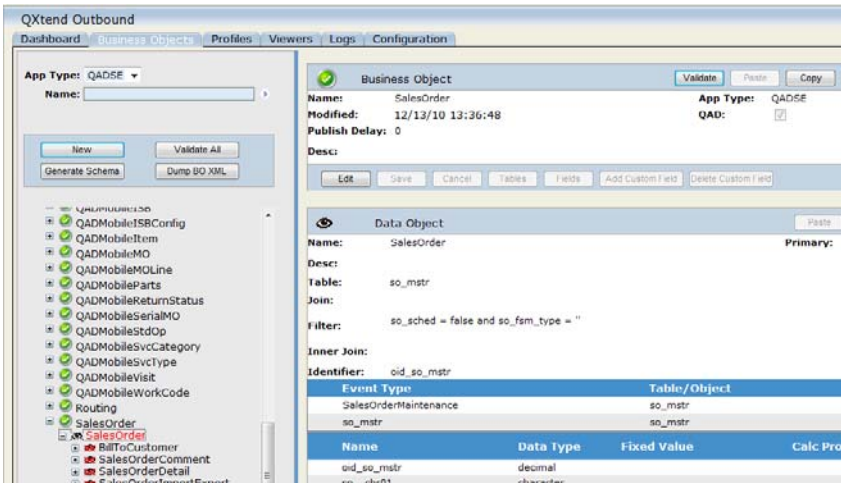


For any data object you received a REVERSE WHOLE-INDEX warning on, determine whether you require refreshed data from an update to the object. For example, on a sales order object, you may not want to refresh data when comments are updated, but do want to when addresses and sales data are changed.

2 Select the data object in the navigation tree to open it.

3 Select the event types you want to be active. When inactive, events in the data object's source application tables are ignored. However, the data for the object is retrieved when events occur in other watched data objects for the business object.

Fig. 3.8
Event Type Activated on the SalesOrderDetail Data Object



4 Click Validate again to verify your changes.

5 Click Save to save the new business object. The screen redisplays with the Edit button available.

You can edit the modified business object at any time including the object name, publish delay, description, and tables.

Creating New Business Objects

You can create new business objects to accommodate your own business requirements. If you do this, you should keep the following considerations in mind.

- QXO is installed with a set of event types. These event types represent a subset of QAD Enterprise Applications tables that already include schema triggers. If you need to reference data in an QAD Enterprise Applications table that is not included in this subset, you must add schema triggers to the table.
- QXO is also installed with trigger files for every standard QAD Enterprise Applications table. Triggers in QAD Enterprise Applications SE and above are capable of sending either the rowid or an OID. Only the subset of these trigger files associated with the standard event types is compiled. If you add triggers to a table, you must also compile the associated trigger files. The schema triggers are usually compiled during installation; however, they must be added to the database schema to take effect. See Step 7 in “Creating a Standard Business Object” on page 76 for details.
- QXO is configured to work with standard QAD Enterprise Applications tables. However, it can be extended to work with business objects based on custom tables or in a side database or a non-QAD Enterprise Applications database. A non-QAD Enterprise Applications database must be Progress based and support the use of triggers. To implement these kinds of business objects requires that you create your own trigger files, since these are not included when QXO is installed.

Note These considerations do not apply to business objects that employ DDP, since direct publishing passes the entire business object to QXO—schema triggers are not relevant.

Instructions are provided here for creating:

- New standard business objects
- Business objects that employ DDP
- Business objects for custom tables or non-QAD Enterprise Applications tables

Creating a Standard Business Object

Use the following steps to create a new business object that references data from standard QAD Enterprise Applications tables.

1 In the left-hand Business Objects frame, click New. The Business Object definition screen displays.

2 Enter values as described in step 3 under “Modifying Business Objects” on page 73.

3 Use the Tables button to add tables to the object. Click OK to save the table definitions.

4 After the new tables appear as data objects in the navigation tree, click on Validate to test the new business object tree.

If you receive warnings, use the instructions in “Validating and Optimizing Business Objects” on page 74 to modify data watch settings.

5 Click Save to save the new object.

6 Ensure that all tables included in the new business object are active event types.

   a If the table is defined as an event type, all you need to do is follow the steps in “Editing Source Application Event Types” on page 29 to activate it.

   b If the table is not already defined as an event type, follow the steps in “Adding Source Application Event Types” on page 30 to add it. Then complete steps 7 and 8.

7 If your business object includes data from one or more tables that are not already in the subset of tables that includes triggers, you must add replication write and replication delete triggers to the QAD Enterprise Applications schema for each table.

   a Locate the following file and open it in a text editor:

   ```
   MfgproInstallDir\qxtend\qxo\progress\mfgpro\src\df
   \delta.df
   ```

   b Remove existing entries.

   c Add an entry for each new table you are activating. The name of the replication write trigger is based on the table name prefix with `rw.t` appended; the delete trigger appends `rd.t`. For example, to activate the abs_mstr table, create the `absrw.t` and `absrd.t` triggers.

   ```
   UPDATE TABLE "abs_mstr"
   TABLE-TRIGGER "REPLICATION-WRITE" NO-OVERRIDE PROCEDURE “absrw.t" CRC "?"
   UPDATE TABLE "abs_mstr"
   TABLE-TRIGGER "REPLICATION-DELETE" NO-OVERRIDE PROCEDURE "absrd.t" CRC "?"
   ```

   d Launch MFG/UTIL and choose Database|Load Database Schema (.df) File.

   e Connect to your QAD Enterprise Applications qaddb database.

   f Select the `delta.df` file you modified with the triggers and choose OK.

8 After adding schema triggers, you must compile the associated trigger files.

   a Create a compile list file named `NewTriggers.wrk` in:

   ```
   QXOMFGsrvInstallDir\qxo\src
   ```

   b List all of the new trigger files that correspond to the triggers you are adding to the schema. In the previous example, the work file would list:

   ```
   absrw.t
   absrd.t
   ```

   c Launch MFG/UTIL and choose Programs|Compile Procedures. Compile the work file specifying the new compile list name.

Creating a DDP Business Object

You can define DDP business objects for source applications regardless of their type (DDP and non-DDP). DDP business objects are indicated by the DDP check box in the Business Object screen.

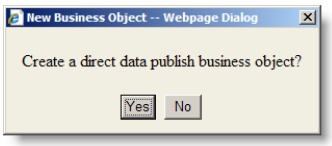
For DDP business objects that have more than one top-level table, one data object must be flagged as the primary data object, by which the raw message will be identified based on the value of the data identifier field of that data object; this is usually tContextInfo. When you load a DDP business object, tContextInfo is marked by default as being the primary data object. This is indicated on the screen by the selection of the Primary check box.

Use the following steps to create a new DDP business object.

1 In the left-hand Business Objects frame, click New. For a non-DDP source application type, the New Business Object box displays.

Note For a DDP source application type, the Load Schema box displays directly.

Fig. 3.9
New Business Object Box



2 Click Yes. The Load Schema box displays.

Fig. 3.10
Load Schema

Schema File. Specify the schema document for the business object.

Note If required, the schema document for the business object can be reimported later by clicking the Reload button at the top of the business object definition screen. The schema may have to be reimported if the source application is modified. Reimporting the schema allows you to retain previous messages.

QAD Financials. Select the field if the schema you are loading is for QAD EE Financials.

Rename Profile Nodes. Select the check box to have the system convert the underscore-delimited names of tables in your business object into camelcase notation when creating the nodes in the associated profile. For details about notation used for elements, see Chapter 24, “QDoc Specifications and Standards,” on page 275.

3 Click Save to save the new object.

Creating a Business Object for Custom Tables

Use the following steps to create a new business object that references data from custom QAD Enterprise Applications tables, a custom side database, or a non-QAD Enterprise Applications Progress database.

1. In the left-hand Business Objects frame, click New. The Business Object definition screen displays.
2. Enter values as described in step 3 under “Modifying Business Objects” on page 73.
3. Use the Tables button to add tables to the object. QAD QXtend reads all the tables from the connected database and displays them for selection. Click OK to save the table definitions.
4. Click Save to save the new object.
5. Create event types for each table to be referenced using the instructions in “Adding Source Application Event Types” on page 30. Ensure that the new event types are active.

   For each table associated with an active event type, add a replication write and replication delete trigger to the schema. Create a `.df` file with an entry for each table you are activating using the instructions in step 7 under “Creating a Standard Business Object” on page 76 as a model.

   For custom QAD Enterprise Applications tables, you can use MFG/UTIL to load the `.df` file. For other applications, use the tools supplied with your application.
6. Trigger files are provided with QXO for all standard QAD Enterprise Applications tables. For non-standard tables, you must create replication write and replication delete trigger files for each affected table. Use the following code samples as a model.

   For QAD SE and above, this is the trigger procedure for replication write for `cm_mstr`:

   ```
   /* RECORD OUTBOUND EVENT. */
   {qxotrig.i
       &TABLE-NAME = 'cm_mstr'
       &ROW-ID = string(rowid(cm_mstr))
       &OID = string(cm_mstr.oid_cm_mstr)
       &TRIGGER-TYPE = 'WRITE'}
   ```

   For QAD SE and above, this is the trigger procedure for replication delete for `cm_mstr`:

   ```
   /* RECORD OUTBOUND EVENT. */
   {qxotrig.i
       &TABLE-NAME = 'cm_mstr'
       &ROW-ID = string(rowid(cm_mstr))
       &OID = string(cm_mstr.oid_cm_mstr)
       &TRIGGER-TYPE = 'DELETE'}
   ```

   The `&DOMAIN-CODE` value is intended for use with QAD Enterprise Applications versions eB2.1 and above, which support separate logical partitions within a database. If your application database can be split into categories similar to QAD Enterprise Applications domains, you may want to use this entry; otherwise, leave it blank.

   If you do decide to use the domain value, you must define the domain to QXO. See “Adding Source Application Domains” on page 27.
7. After adding schema triggers and creating the corresponding trigger code, you must compile the code using your application compiler. For QAD Enterprise Applications custom tables, you can use MFG/UTIL.
8. Add the trigger file directory to the application PROPATH so that the code can be executed.

Deleting Business Objects

To delete a business object you created, follow these steps. This process deletes the business object and any profiles that are based on it. You cannot delete QAD-supplied business objects.

1 Select the business object from the navigation tree. The business object displays. The Edit, Copy, and Delete buttons are active.

2 Click Delete to remove the profile. You are prompted to confirm.

3 Click OK to continue.

Modifying Business Object Data Objects

Once you have created or modified a business object, you can modify the related data objects.

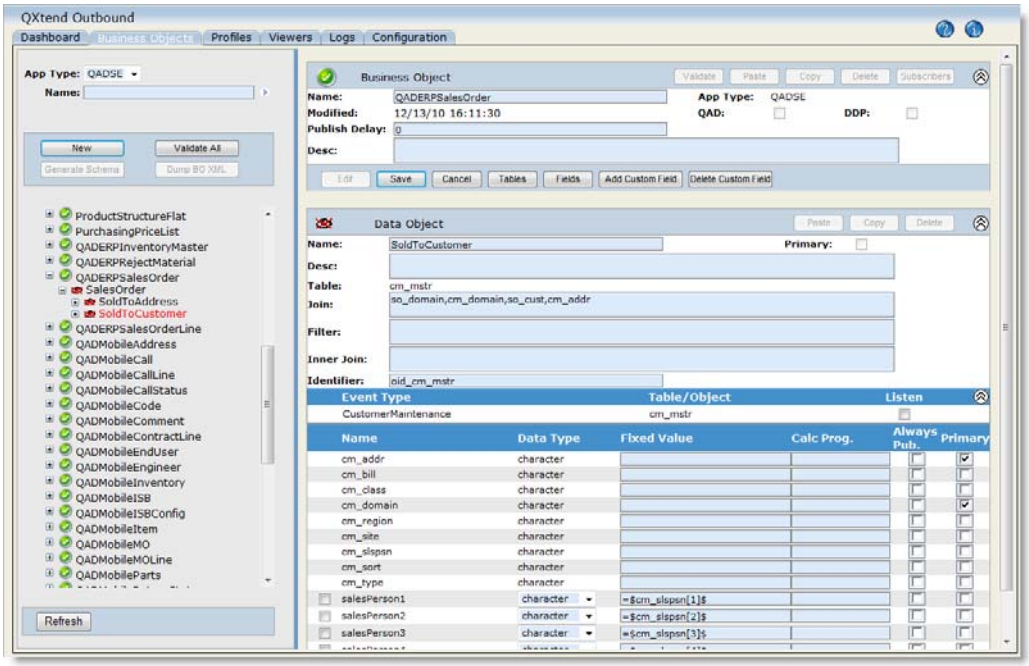
Note Business objects that employ DDP cannot be modified, except for those values that are enabled on screen during edit mode. These attributes are indicated below.

1 In the object tree on the left, open the new business object and select a table within the tree. The data object details for that table display in the right-hand frame.

If you select the top-level table, no join information displays. If you select one of the lower-level tables as shown in Figure 3.11, the join information displays.

2 Click Edit in the Business Object frame at the top left. The editable fields in the data object become active.

Fig. 3.11
Table Data Object



Enter values for the data object as follows:

Name. Enter a name (maximum 35 characters) for this data object. Data object names must be unique within a business object.

Table. The system displays the application table name.

Description. Enter a brief description (maximum 30 characters).

This attribute may be modified for business objects that employ DDP.

Join. Enter a comma-separated list of join field pairs, each of which is also comma-delimited. Each pair contains the parent table join field, then the child table join field.

The joins are used to navigate upward to the top of the business-object hierarchy when processing application events.

Filter. Optionally, enter filter criteria to limit the data stored after extraction from the application. The filter uses Progress 4GL expressions.

Figure 3.12 shows a relatively complex join and filter designed to join the abs_mstr and sod_det using the fields abs_order and sod_nbr, and then limiting or filtering the data extracted to those records where the sod_line value is equal to the abs_line value as converted to an integer.

Fig. 3.12
Complex Join and Filter

Inner Join. Specify the inner join query you want to use. An inner join requires each record in the joined tables to have a matching record in order for the join to be successful. For details about using inner joins, see “Using Inner Joins in Data Objects” on page 83.

Identifier. Specify the field in the data object that is the unique identifier. This field is used to match the current event message object with the previous object in order to determine what changes may have occurred. This field must exist on the data object and be the only field on a unique index. The identifier must be specified if the Use Rowids check box for the source application type is not selected. For details see “QXO Source Applications” on page 23.

Important The identifier must always be specified if the source application type does not use rowids to identify events. For application types QAD SE and QAD EE the identifier defaults to the OID field of the table, which uses the pattern oid_<*tablename*>—for example, oid_ad_mstr.

3 You can modify the behavior of individual fields in the table. Each field in the table displays in a table with active fields. Modify the values for each field as follows:

Data Type. If it is a custom field, select from one of the following data types: character (default), date, datetime, datetime-tz, decimal, int64, integer, and logical; if it is a database field, the data type is populated by the system and is read-only.

Fixed Value. Optionally, specify a fixed value that you always want 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.

For example, if a profile data object contains two nodes—field1 and field2— and the fixed value component of field2 is =field1, field2 will be populated with the value of field1. If field1 does not exist, the string =field1 will be the value of field2.

Fixed value fields can also contain Progress expressions and perform built-in Progress calculations; for example, `=today+2`. Fields from the current buffer can be referenced in the current calculation by including the XML node name surrounded by dollar ($) characters; the expression must start with an equals sign (=) character. For example, `= substring($ptDesc1$,1,12)`.

The system determines node values in this order: a fixed value, a calculated value, the value supplied in the event message.

You can double-click in the Fixed Value field and view or enter data in a pop-up window.

Fixed values are typically used with custom fields, described in step 5.

The fixed value you provide must be consistent with the data type you specified for the field. If there is a mismatch, the system displays an error message when you try to save the data object.

Calc Program. Optionally, enter the name of a Progress program (`.p`) to be 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. For details on calculated field programs see “Implementing Calculated Fields” on page 63.

Make sure the calculated value will be consistent with the data type you specified for the field. If there is a mismatch, the system does not prevent you from saving the data object, but will run into errors during data publishing.

Note The Progress program does not necessarily have to run on the AppServer.

The system determines node values in this order: a fixed value, a calculated value, the value supplied in the event message. Calculated values are typically used with custom fields.

Note Fixed values and calculated programs can be defined on the business object, the associated profile, or both. If you define a fixed value or calculated program on the *business object*, all profiles associated with that business object will have that fixed value or calculated result. Defining a fixed value or calculated program on a business object means that associated profiles can filter on that value or result. If you define a fixed value or calculated program on the *profile*, only that profile has that fixed value or calculated result.

Always Publish. Indicate if the field should be published whenever the data object that contains it is published, regardless of whether the field was changed by the application event.

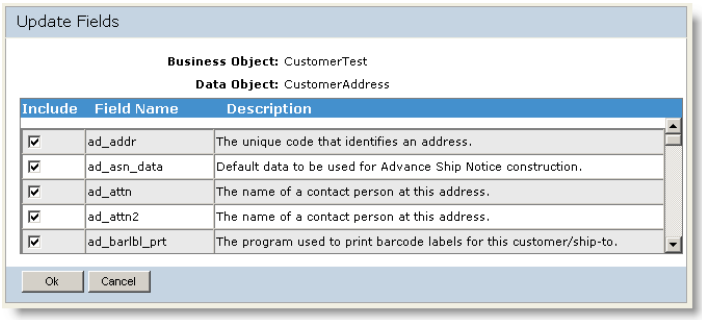
This attribute may be modified for business objects that employ DDP.

Primary. Indicate if this field is part of the primary index for the table associated with this data object. For QAD-supplied business objects, the primary fields are already set.

This attribute may be modified for business objects that employ DDP.

4 Buttons in the Business Object frame allow additional configuration of the data object. Click Fields to display the Update Fields screen.

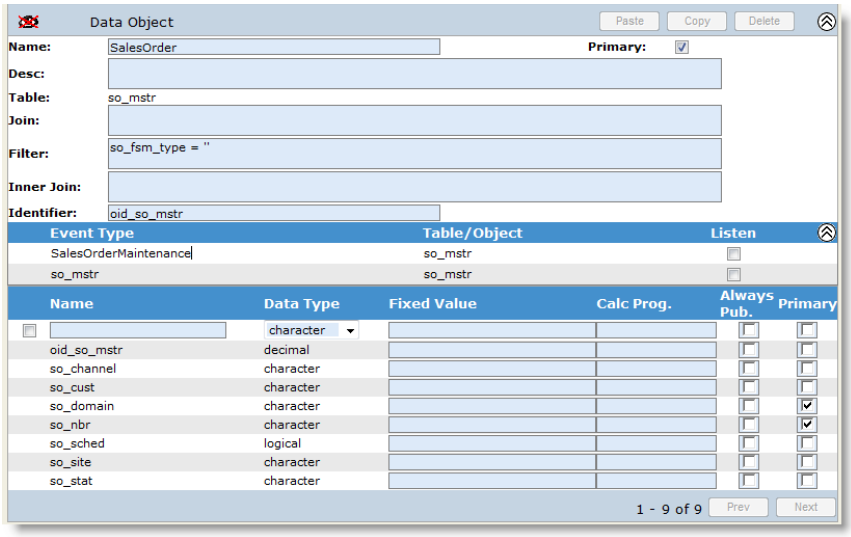
Fig. 3.13
Update Data Object Fields



By default all fields in a QAD-supplied business object are included. To exclude a field from the business object, clear the Include check box for the field.

5 Click Add Custom Field to add a new field to the data object.

Fig. 3.14
Adding a Custom Field



Enter a field name, specify a data type, and add a fixed value or Progress calculation program if needed.

To delete a custom field, click the check box to the left of the field and click Delete Custom Field.

6 Click Save to save the data object and any changes you have made.

Using Inner Joins in Data Objects

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.

For example, you could create an inner join query to extract only those sales orders that have line items:

```
first sod_det where
    so_nbr=sod_nbr
    so_domain=sod_domain
```

You use inner joins to query fields and tables in other business objects in your system that contain other types of data you require—address information, for example—not related to the current data object.

```
first ad_mstr where
    ad_addr=so_cust and
    ad_ctry="US" and
    ad_domain=so_domain
```

You combine multiple inner join queries by using a comma to separate the queries, as shown here:

```
first sod_det where
    so_nbr=sod_nbr
    so_domain=sod_domain,
first ad_mstr where
    ad_addr=so_cust and
    ad_ctry="US" and
    ad_domain=so_domain
```

In this example, the query first searches the sod_det table in the current business object and identifies those sales orders that have line items. Then the query searches the `ad_mstr` table—this table could be in a different data object in a different domain—and returns the rows that have customers that are located in the US. When specifying inner joins, all query conditions you specify must be true in order for the query to extract data.

Note If you want to use the Query Service on business objects that have filters and/or inner joins as part of their definition, you can control whether these conditions remain active in the request QDoc by setting the `IgnoreBOFilter` and `IgnoreBOInnerJoin` nodes as required. These nodes take logical values. For details about the Query Service, see Chapter 10, “QXtend Query Service,” on page 123.

Copying or Deleting Data Objects

You can copy a data object from one business object and paste it into another. This eliminates configuration steps where multiple business objects such as sales orders, invoice history, and customer all share a data object such as addresses. You can also delete data objects if they were copied into the wrong business object.

Note Data objects cannot be copied to or deleted from business objects that employ DDP.

1. While the source business object is in edit mode, select the source data object in the left-hand navigation tree.
2. In the Data Object frame, click Copy.
3. Use the navigation tree to open the target business object, and data object, as necessary.
4. In the target business object Data Object frame, click Paste. The copied data object is pasted in.
5. To delete a data object from a business object, open the data object using the navigation tree. Click Delete in the Data Object frame.

Managing Business Object XML Files

Two additional business object functions let you generate an XML schema from the business object in order to validate the XML structure, and dump the business object and associated profiles in an XML format that can be loaded into another instance of QXO.

Navigate to the business object you want to work with. In the navigation tree frame click:

Validate All. Validate all business objects under the selected source application type.

Note If there are many business objects, the validation process may take a few minutes.

Generate Schema. Generate the XML schema from the business object in order to validate its structure.

Dump BO XML. Create a file with the XML representation of the business object and its associated profiles. These files can be loaded into another instance of QXO using the XML Load utility on the Configuration tab. See “Importing Business Objects and Profiles” on page 56. Please note that for DDP business objects, the system dumps XSD schema files as well as XML files.

The dump file is created with the following naming convention:

`QXtendInstallDir\boXML\AppType\BOName\ProfileName`

Chapter 4

QXtend Outbound Profiles

The following material covers implementation of QXtend Outbound (QXO) profiles.

Introduction

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.

Besides the default profile, some business objects may also have default data synchronization profiles supplied by QAD. These data synchronization profiles are used for synchronization of data between different QAD Enterprise Applications domains or instances. A data synchronization profile usually has the same name as that of the corresponding QDoc used on the QXtend Inbound side and also contains the QDoc version information. Default data synchronization profiles typically use UIAPI QDocs. In QAD EE, if there is a SIAPI QDoc available for the business object, then the default profile uses this SIAPI because of its better performance.

Profiles can be triggered to be published only when certain event types were the original trigger for the raw message.

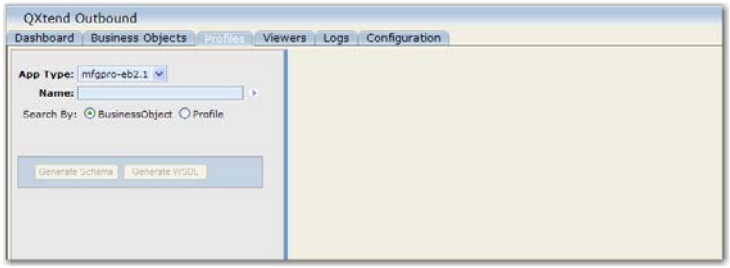
Important New customers to QXtend will always use QDoc Version 1.1. QDoc Version 1.0 is described for backward compatibility reasons.

Viewing QXO Profiles

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 screen illustrated in Figure 4.1 displays.

Fig. 4.1
Profile Screen



App Type. Select the source application type from the drop-down list. Source applications are defined in the Configuration tab. See “QXO Source Applications” on page 23. The last selected value is saved and used as the default entry the next time this screen is viewed.

Name. Use the Name field to find profiles to view or update. Click Business Object to search by business object names; click Profile to search by profile names. See “Filtering Names” on page 20.

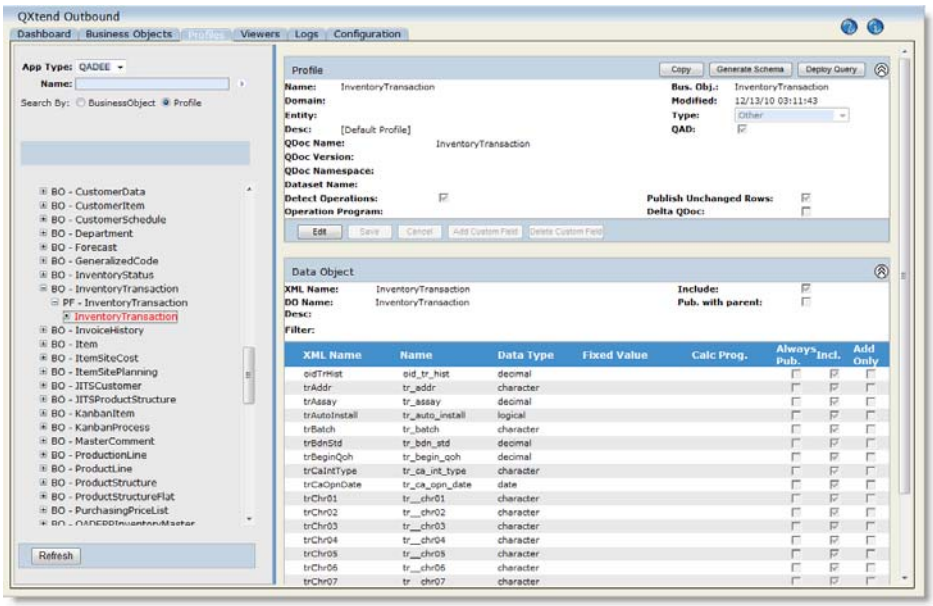
Click the arrow next to Name to display all available business objects with the Name field left blank, or to search within the available business objects for objects that match your Name entry.

Modifying Profiles

Since you cannot edit the QAD-supplied profiles or create a new one, you must copy an existing profile and modify it. To create a copy and modify a profile, follow these steps.

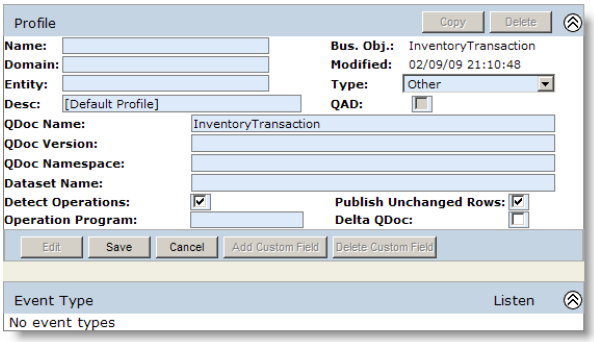
1. Select the source profile from the navigation tree. If it is a QAD-supplied object, the only button active is Copy.
2. Open the nodes in the profile in the navigation tree to view the data objects in the profile.

Fig. 4.2
Profile Copy



3. Click Copy. The new profile is created including all related data objects. The new profile definition screen displays. A new name is provided for the profile by default.

Fig. 4.3
Profile Definition



4 Enter values for the new profile as follows:

Name. Update the name as required.

Business Object. The business object this profile is based on displays in this field.

Domain. For eB2.1 or above databases, specify the domain associated with the profile. If you leave this field blank in eB2.1 or above, the profile applies to all domains in the database. If a domain value is specified, the profile only sends QDocs for events extracted from the specified domain.

For other application types, you can use domain according to your own business requirements or leave it blank.

Entity. Specify the entity you want to create the QDoc for.

If you leave the field blank, the profile will create the QDoc for all entities.

Modified. No edit allowed. This displays the date and time this profile was last saved.

Type. Choose a profile type. The profile type identifies the response type so that the response can be parsed correctly.

Desc. Enter a description of the object. Consider including the creation date, object author, and purpose.

QAD. Indicates a QAD-standard profile.

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.

QDoc Version. Use this to distinguish between QAD Enterprise Applications QDoc versions. This is the version designation that appears in the QDoc headers and is one of the values required to validate the QDoc.

QDoc Namespace. If using the QDoc XML Syntax setting of Other, specify a URL for QDocs generated using this profile. You do not have to specify a namespace if using an XML Syntax setting of 1.1 or 1.0. Like the QDoc version, the QDoc namespace is a required value in the QDoc headers to validate the source of the QDoc.

Dataset Name. Specify the name of the root node in the QDoc XML.

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.

Note For details about customizing operation programs, see “Operation Programs” on page 402.

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.

If you use the Add Only option to avoid overwriting fields that are locally owned by the target application, you must enable the Detect Operations option.

The archive service will not archive subscriber messages that are required to correctly determine the operation of future subscriber messages.

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.

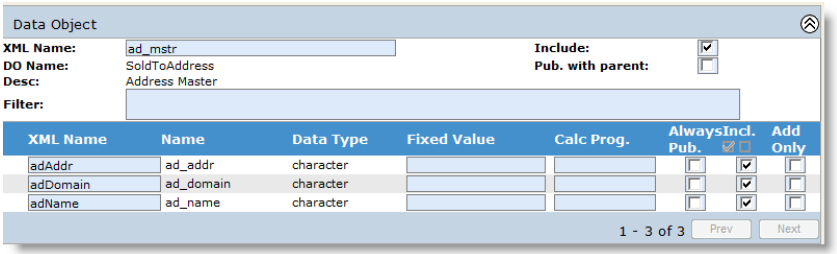
Example You have a sales order consisting of 20 line items, and the price of one line item changes. Select Publish Unchanged Rows to publish the changed row and unchanged rows. To publish only the changed row, clear the check box.

Operation Program. Enter the name of the operation program to use with this profile. This option is used in conjunction with the Detect Operation check box. The operation program you enter must be a Progress (.p) program contained in the PROPATH.

Note Only select this check box if you want to perform an operation calculation, or if you want the operation value to be entered into a different field than the operation node. For details about using operation programs in profiles, see Appendix B, “Parameter Data,” on page 401.

5 Click Add Custom Field to add a new field to the data object.

Fig. 4.4
Adding a Custom Profile Field



Enter a field name, specify a data type, and add a fixed value or Progress calculation program if needed. The field name must contain only alphanumeric characters; the name is validated to ensure no illegal characters are included and that the field name is a valid XML node.

To delete a custom field, click in the selection box to the left of the field and click Delete Custom Field.

Event Types. Select or clear the check boxes for event types as required to determine which events must occur in order for the profile to be published. 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.

6 Click Save to save the new profile and any changes you have made.

Deleting Profiles

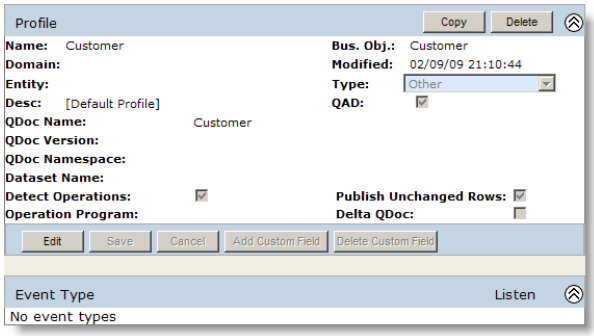
To delete a profile you have copied and modified, follow these steps. This process deletes the profile and the data objects stored within it. You cannot delete the QAD profiles or data objects.

If you delete all user-configured profiles associated with a business object, a new default profile is generated. This occurs because a business object cannot exist without a profile. In this scenario, the delete essentially refreshes the default profile for the business object.

In addition, if a QDoc does not have an associated profile because the profile has been deleted, the profile name displays as UNAVAILABLE in the various reports and views.

1 Select the profile from the navigation tree. The profile displays. The Edit, Copy, and Delete buttons are active.

Fig. 4.5
Profile Header



2 Click Delete to remove the profile. You are prompted to confirm.

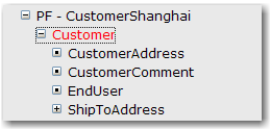
3 Click OK to continue.

Modifying Profile Data Objects

Once you have created a copy of a profile and saved it, you can modify the data objects within the profile.

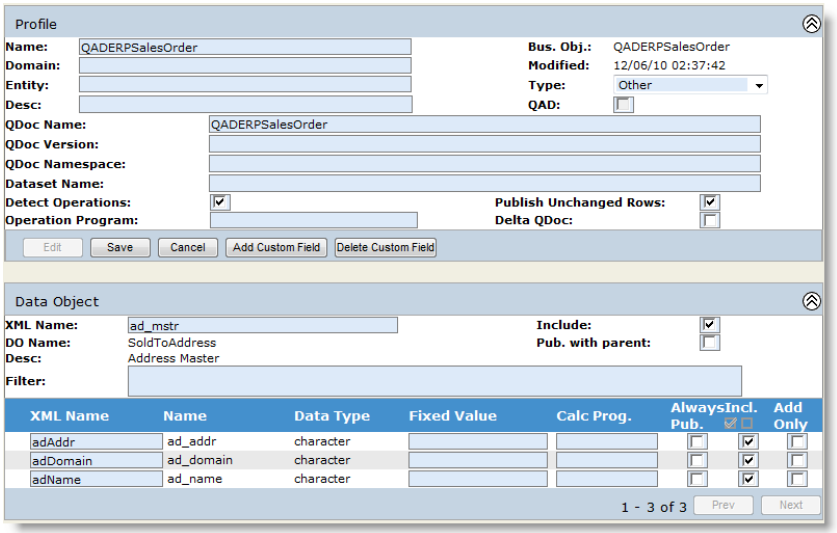
1 Select a non-QAD profile in the navigation tree and open the data object you want to modify or delete.

Fig. 4.6
Profile Data Object Nodes



2 The data object displays in the right pane. Click Edit to access the profile and data object fields.

Fig. 4.7
Profile and Data Object Definition



3 Use the descriptions in step 4 in “Modifying Profiles” on page 89 to modify the profile settings. Use the following field descriptions to modify the data object.

XML Name. The name of the XML element or node for the data object in published QDocs.

Include. Indicate if you want this data object 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.

DO Name. The system displays the name given this data object in the business object used as a template.

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.

The hierarchical display in the navigation tree shows the parent/child relationship between data objects.

Note This option will only work if a parent record has only one child record. If a parent record has multiple child records, when the child records are modified the changes might not appear in the generated QDoc as the QDoc will only contain data for the first child record.

Description. Enter a brief description.

Filter. Optionally, enter filter criteria to limit the data extracted from the application. The filter uses Progress 4GL expressions.

4 Below the profile data object, you can modify settings related to each field in the profile.

XML Name. Enter the XML element node name used to publish this field in outbound messages.

Data Type. If it is a custom field, select from one of the following data types: character (default), date, datetime, datetime-tz, decimal, int64, integer, and logical; if it is a database field, the data type is populated by the system and is read-only.

Fixed Value. Optionally, specify a fixed value that you always want 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.

For example, if a profile data object contains two nodes—field1 and field2— and the fixed value component of field2 is =field1, field2 will be populated with the value of field1. If field1 does not exist, the string =field1 will be the value of field2.

Fixed value fields can also contain Progress expressions and perform built-in Progress calculations; for example, `=today+2`. Fields from the current buffer can be referenced in the current calculation by including the XML node name surrounded by dollar ($) characters; the expression must start with an equals sign (=) character. For example, `= substring($ptDesc1$,1,12)`.

The system determines node values in this order: a fixed value, a calculated value, the value supplied in the event message.

Fixed values are typically used with custom fields.

The fixed value you provide must be consistent with the data type you specified for the field. If there is a mismatch, the system displays an error message when you try to save the data object.

Calc Program. Optionally, enter the name of a Progress program (`.p`) to be 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. For details on calculated field programs see “Implementing Calculated Fields” on page 63.

The system determines node values in this order: a fixed value, a calculated value, the value supplied in the event message. Calculated values are typically used with custom fields.

Make sure the calculated value will be consistent with the data type you specified for the field. If there is a mismatch, the system does not prevent you from saving the data object, but will run into errors during data publishing.

Note Fixed values and calculated programs can be defined on the business object, the associated profile, or both. If you define a fixed value or calculated program on the *business object*, all profiles associated with that business object will have that fixed value or calculated result. Defining a fixed value or calculated program on a business object means that associated profiles can filter on that value or result. If you define a fixed value or calculated program on the *profile*, only that profile has that fixed value or calculated result.

Always Publish. Indicate if the field should be published whenever the data object that contains it is published, regardless of whether the field was changed by the application event.

Include. Indicate if you want this field included in messages generated based on this profile.

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.

For example, you might use the Add Only check box in a data synchronization scenario where you have fields that are maintained in the local domain and not replicated from the master domain, except in the case where the record is being created. Selecting the Add Only check box means the fields are published only when the record is being added.

5 Click Save to save the data object changes.

Managing Profile XML Files

An additional profile function lets you generate an XML schema from the profile in order to validate the XML structure.

1 In the navigation tree, select the profile you want to work with.

2 Click Generate Schema. This generates the XML schema from the profile.

Chapter 5

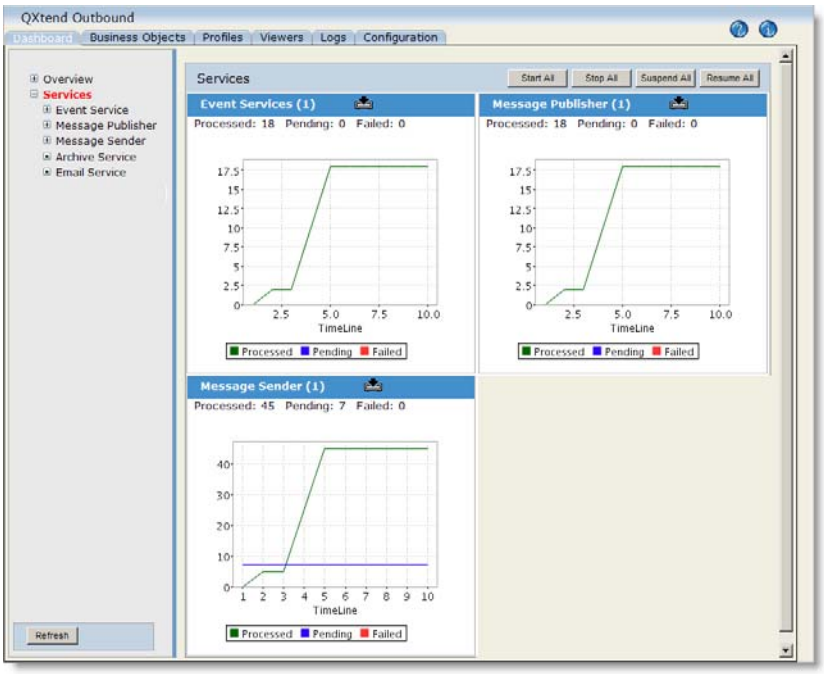
QXtend Outbound Dashboard

The following material covers the QXtend Outbound (QXO) dashboard.

Introduction

The Dashboard displays by default with a system overview. There are two basic views—Overview and Services—displayed in the navigation tree. Click Overview to view the output from your defined source applications and subscribers. Click Services to view details for event services, message publishers, message senders, and archive services.

Fig. 5.1
Services Dashboard Showing the Addition of a Customer in a Source Application



Note The input date format on the dashboard UI is American and is not configurable. Database date formats can be either dmy or mdy, but both the `qxevents` and the `qxodb` databases must have the same format. Note also that time data being returned from the database, such as Date-Time fields on the Logs screen or modification time fields on the profiles, are in the format of the `qxodb` database.

Use the navigation tree on the left to drill down to further details. The charts that display for each source application, service, and subscriber view use a standard timeline format showing the previous hour of activity broken into ten six-minute periods. Adjust the view by resetting the start and finish times in the individual views. The minimum total period is 10 minutes.

Note The archive service displays archive status information as a table, not a chart.

Click the Download icon in the title bar of a chart to view the data represented in the chart in an HTML table. You can then save this data for import to a spreadsheet application such as Excel and create your own graphs.

Fig. 5.2
Download Icon

Note in Figure 5.1 the numbers at the top of each chart. One event service, one message publisher, and one message sender are running. Three subscribers are active. The event service processed three events resulting from adding a customer record. The message publisher processed these events as a single message. The message sender then created three QDocs to meet the needs of the three different subscribers.

Viewing Services Summaries

When you select one of the top-level services nodes—Event Service, Message Publisher, and so on—a summary of all the services of that type displays. In the table at the top of the screen, each instance of the service and its status are displayed. The possible statuses are:

STRT. The instance is starting. Blue.

STOP. The instance is stopped. You can restart the instance using the Start button at the bottom of the screen. Red.

IDLE. The instance is idle. Green.

PROC. The instance is processing an event or a message. Yellow.

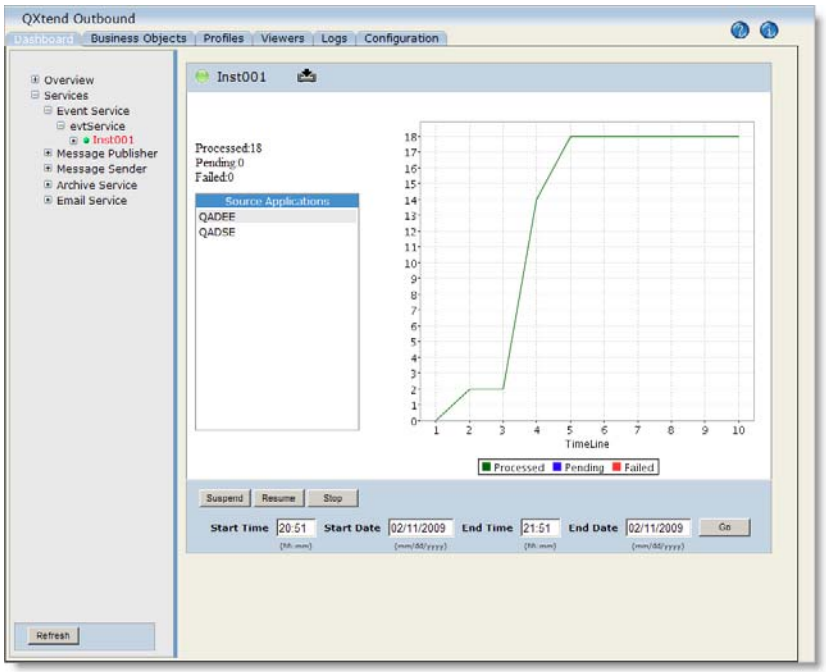
SUSP. The instance is suspended. You can resume the instance using the Resume button at the bottom of the screen. Light gray.

DEAD. The instance is dead. You can replace the instance by creating a new one. Dark gray.

Viewing Specific Services

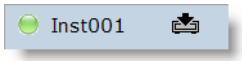
Figure 5.3 shows how the Dashboard looks after you have drilled down into the event services view. The views for message publishers, senders, and subscribers are identical, except that subscribers do not require the buttons displayed in the other screens. The archive service and email service display status information as a table, not a graph.

Fig. 5.3
Event Services Dashboard



The status of the individual service instance displays as a round, colored icon next to the instance name. See “Viewing Services Summaries” on page 99 for a summary of the color codes.

Fig. 5.4
Status Color Code for Services Instance



You can use the buttons to perform the following actions:

Start. Start all services if not already started.

Stop. Stop all services if not already stopped.

Suspend. Suspend all services if not already suspended.

Resume. Resume all services that have been suspended.

Add. Add a new instance of the service based on this session profile.

Important If an active event type is not registered with any event service, it still generates events in the qxevents database but they will never be processed. To prevent unwanted events from accumulating in the qxevents database, the system stops any event service that has registered source application with active event types not registered with any event service. When you start an event service, the system checks whether all the active event types within each of its registered source applications are registered with an event service. If not, the system shuts down the just started event service and displays a warning message. For information on event type registration, see “Registering Event Types with an Event Service” on page 35.

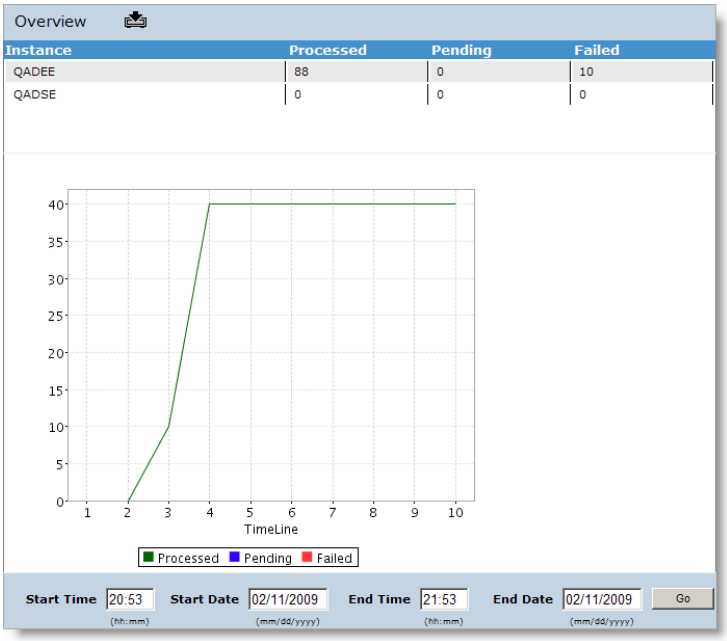
Use the start and end time fields to limit the data displayed in the chart.

You also can control services from the command line; for details, see “Session Control Tool” on page 115.

Monitoring Source Applications

Under the Overview node, you can monitor your source applications. Each source application is listed under the Overview node. Click the Overview link to see a summary of all source application activity.

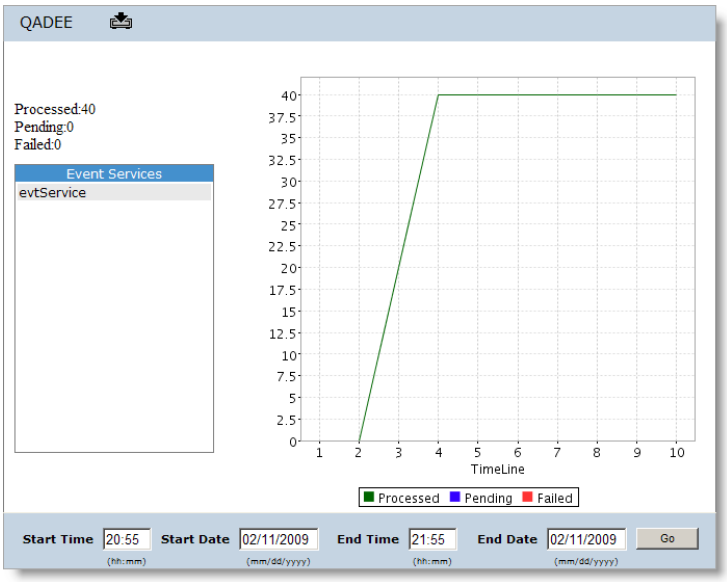
Fig. 5.5
Source Application Overview



The table at the top shows the individual source applications. Instance is the source application name. Processed, Pending, and Failed shows the number of events in each stage. The chart shows the cumulative processing for all source applications.

Open the Overview node in the navigation tree and click any one of the source applications to view specifics.

Fig. 5.6
Individual Source Application View



The event services associated with the source application appear in the table to the left.

Chapter 6

QXtend Outbound Viewers

The following material covers the QXtend Outbound (QXO) viewers.

Introduction

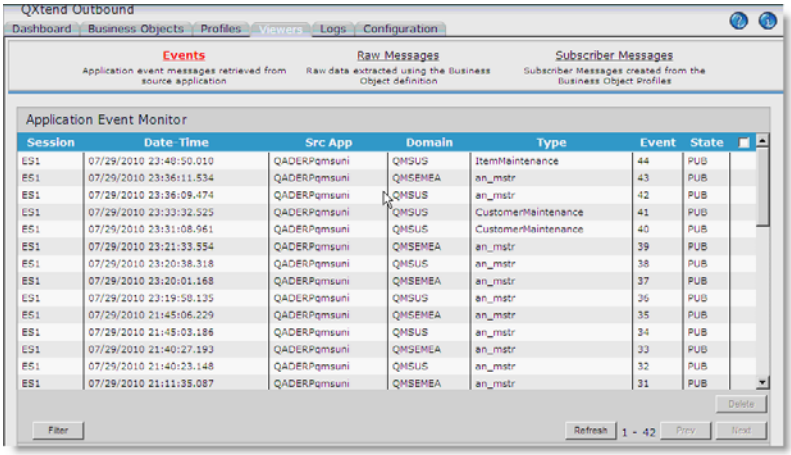
Use the Viewers tab to view:

- Messages generated when QXO successfully reads an event from the `qxevents` database
- Raw data extracted from a source application for all profiles that are monitoring that data
- QDocs created from the business object profiles

Information displays in filterable tables. Each view displays the 50 most recent records. Next and Previous buttons move up and down through the records. If alerts occur, you can display and filter them also. See “Using Table Filters” on page 21 for details on using filters.

Application Event Viewer

Fig. 6.1
Application Event Viewer



Delete. Use the Delete button to delete application event messages with state ERR.

When application data being monitored by QXO is changed, a record is written to the `qxevents` database and queued for processing by the event service. The event service polls the `qxevents` database for new events and when it finds one, it extracts the related application data from QAD Enterprise Applications and saves it in the `qxodb` database in the form of raw data.

A message is recorded each time an event is processed. The message indicates the time the event was processed, any errors that occurred, the source application that caused the event, and other pertinent information.

The application event viewer does not show messages for business objects that employ DDP since these objects are not processed by the event service. However, messages for these business objects are recorded in the raw message viewer.

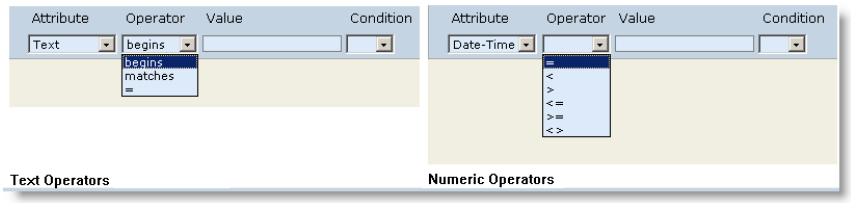
Filtering View Output

To filter the output of a view, use the following steps:

1. Under the Viewers tab, select the view you want to filter from the menu along the top. The full view displays. See also “Using Table Filters” on page 21.
2. Click Filter to open the filter panel.
3. In the filter panel, enter a filter Attribute—session, type, date-time, and so forth as displayed across the top of the viewer—an Operator, and the Value you want to filter by.

   There are two different sets of operators possible, one for numeric values, the other for text as shown in Figure 6.2.

Fig. 6.2
Text Operators



4. If you want to search by multiple values, enter a Condition and click Add. The next filter line displays.
5. Click Run to filter the viewer.

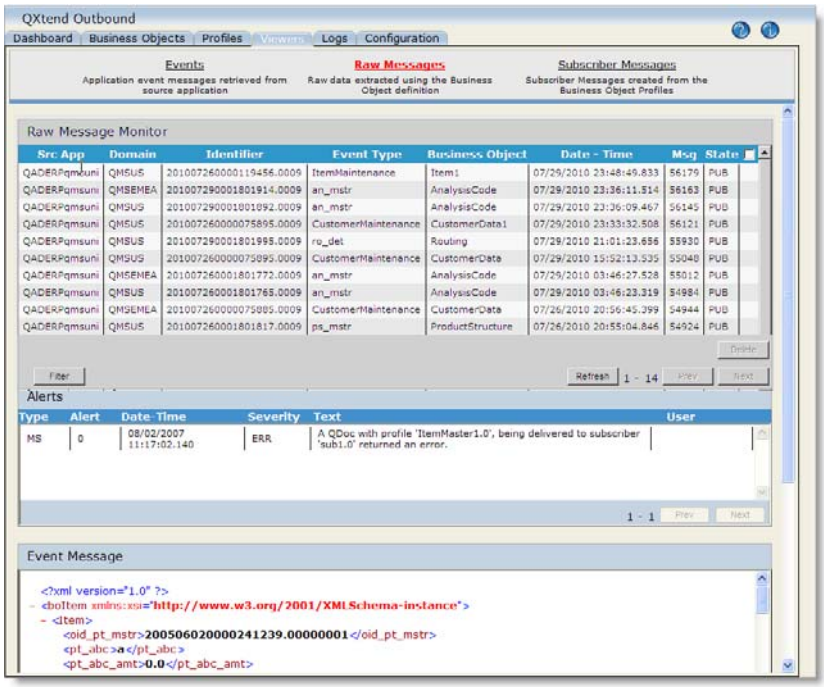
   Note If you change the filter values, then click Next or Previous before clicking Run, your filter changes are lost.
6. Click Delete or Reset to close the filter pane.

Raw Message Viewer

The raw message viewer displays information about the messages stored in the `qxodb` database by the event services. Click any message to view associated alerts and the message content.

The raw message viewer also displays information about business objects that employ DDP, which are passed directly to the `qxodb` database by the source application.

Fig. 6.3
Raw Message Viewer

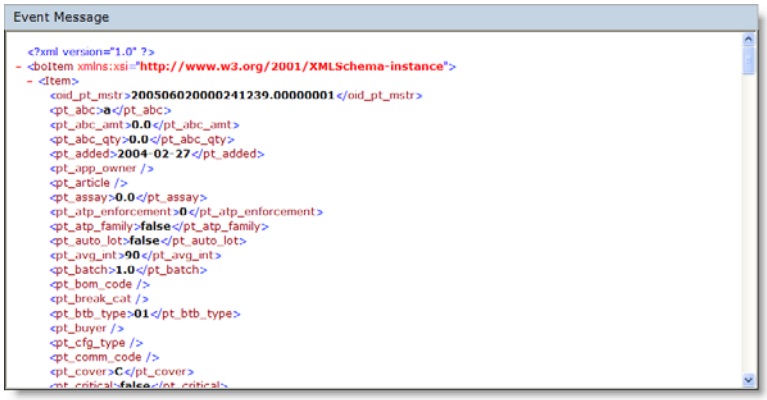


Delete. Use the Delete button to delete extracted data with state ERR.

Note Message Publisher automatically deletes from the `qxodb` database any raw messages from which no profile messages are created.

Scroll down to view the raw message content.

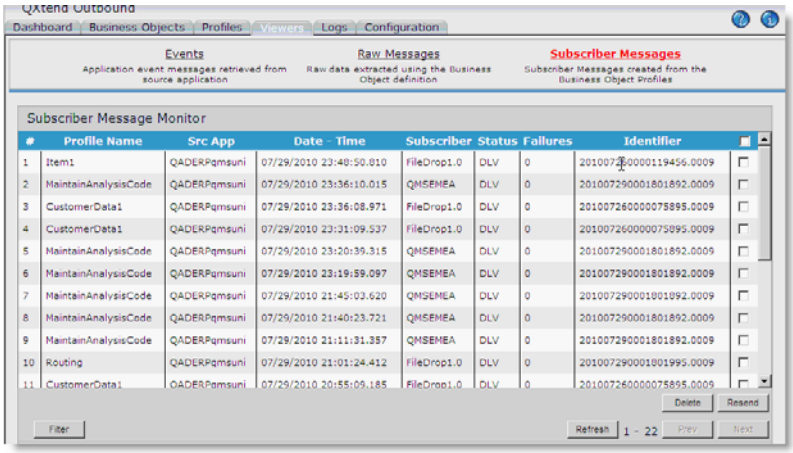
Fig. 6.4
Raw Message Content



Subscriber Messages Viewer

The subscriber messages viewer displays subscriber messages created from business object profiles. Click any subscriber message to view any associated alerts and the raw XML of the request.

Fig. 6.5
Subscriber Messages Viewer



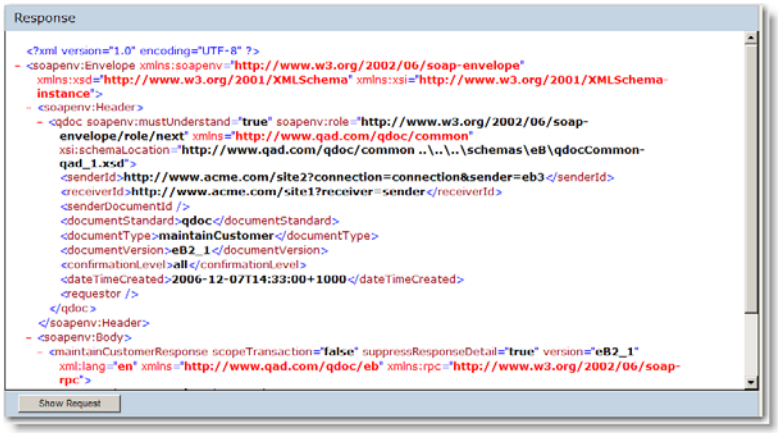
Identifier. The Identifier column uniquely identifies a subscriber message. Identical message identifiers indicate the same subscriber message processed multiple times.

Delete. Use the Delete button to remove a QDoc from the viewer. This also deletes the QDoc record in the `qxodb` database.

Resend. Use Resend to forward a QDoc to any subscribers that are registered for the QDoc profile. A resend changes the status to PEND. After delivery, the status changes to ERR or OK depending on the success of the delivery.

Scroll down to view the QDoc.

Fig. 6.6
Response



Show Response. Use the Show Response button at the bottom of the viewer to view the response returned from the subscriber. Using this button you can view the response to a QDoc if you cannot access the QXtend Message Monitor in the .NET UI.

Chapter 7

QXtend Outbound Log Monitor

The following material covers the QXtend Outbound (QXO) logs.

Viewing QXO Logs and Exceptions

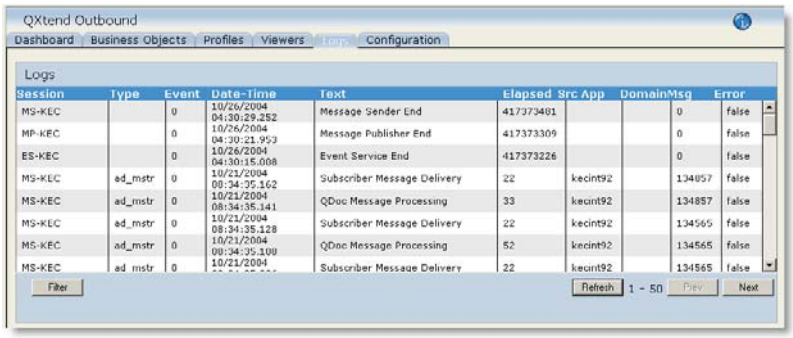
Use the Log tab to view log and alert messages for all transactions in the QXO system and filter them by various criteria.

Messages are written to the log by all of the QXO services when data is extracted and when QDocs are created and published.

You can delete and optionally save all alerts and logs generated by the system by running the archive routine. See “Data Archive” on page 57.

Figure 7.1 illustrates the Logs portion of this screen.

Fig. 7.1
Log Tab



The system returns 50 records. If more than 50 records exist, use the next and previous buttons to display the next group of 50 records. Use the scroll bar to scroll up and down within a group of 50 records. Click Refresh to find new records created since the initial screen display.

Use the Filter button to toggle the display of the filter criteria. See “Filtering View Output” on page 105.

Note Logs and alerts remain in the `qxodb` database indefinitely until they are cleared by running the archive routine. For details see “Data Archive” on page 57.

Additional Logs

For further troubleshooting, review the logs located in *`QXOsrvInstallDir`*`\logs`. The log naming convention for the various services is `session-`*`<numeric-instance-segment>`*`-`*`<service name>`*`.log`. For example, an event service named ES1 may have two agents, Inst001 and Inst002. The log files would be in `001-ES1.log` and `002.ES1.log`.

There are no subscriber logs. Troubleshooting subscriber problems should be done on the subscriber side.

To control log detail, you can edit your start-sess.sh or start-sess.bat file. Open the file in a text editor, locate the LOGLEVEL entry, and change the logging level as desired:

0 Only level 0 messages and one required level 1 message displays.

1 Level 0 and 1 messages display.

2 Level 0, 1, and 2 messages display.

By default this value is set to 0. This is the recommended setting. If you are having trouble debugging an issue, you can change the logging level to display additional errors and messages.

The AppServer broker and server logs, if set up as described during installation, will be named `QXOSession.broker.log` and `QXOSession.server.log`, respectively. Refer to *Installation Guide: QAD QXtend*.

Chapter 8

QXtend Outbound Tools

The following material describes QXtend Outbound (QXO) tools.

Introduction

Various tools are provided with QXO that are used in conjunction with the QXO application. These tools are run from the command line and must be adapted to operate in your environment before they are run.

QXO has the following tools:

- Mass Rowid Synchronization Tool
- Session Control Tool

Mass Rowid Synchronization Tool

From time to time you may need to perform Progress database dump and load procedures for various maintenance-related reasons. Performing dump and load procedures changes the assigned rowids in the QAD Enterprise Applications database; this means that they will no longer be synchronized with the original rowids stored in the QXO database. This reassignment of rowids will result in erroneous messages being generated by QXO.

Note You only need to use this tool if you use rowids—synchronizing rowids is obviously not an issue with OIDs.

To correct the rowid mismatch and resolve this issue, after performing a Progress database dump and load you must run the QXO mass rowid synchronization utility, which is located in the `<qxo>src/tools` directory.

This utility iterates through the records in the Event Message section table of `qxodb` and queries the QAD Enterprise Applications database to determine if the rowids match. If there is a mismatch, the utility synchronizes the Event Message section record rowid to that of the QAD Enterprise Applications rowid.

The utility comprises the following files:

- Startup script (`start.msync`)
- Database parameter (`dbparm.pf`)
- Mass rowid sync program (`rowid-msync.p`)
- Generated log file (`rowidUpdate-msync.txt`)

You run the synchronization utility from the command line. Before running the utility you must:

1. Edit `start.msync` to set the correct values for `DLC`, `PROPATH`, and `MFGDB`.
2. Edit `dbparm.pf` to set the correct values for the databases being used.

The utility creates an entry in the log file in the `<qxo>/src/tools` directory for the first record and for multiples of 100,000 records. You can check this log file periodically to see the progress of the utility.

Session Control Tool

You 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.

Note You also can start and stop services from the QXO Dashboard; for details see “Viewing Specific Services” on page 99.

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

Note The `sess-control.sh` file contains settings for environment variables and database connections. This file is configured according to the information you provide during installation. However, you can manually edit this file to modify these settings as required.

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

```
./sess-control.sh {QUERY|START|STOP|SUSPEND|RESUME} [ALL] [RESUME_SUB]
```

```
./sess-control.sh {QUERY|START|STOP|SUSPEND|RESUME} SERVICES
[RESUME_SUB]
```

```
./sess-control.sh {QUERY|START|STOP|SUSPEND|RESUME} ES [event service
code]
```

```
./sess-control.sh {QUERY|START|STOP|SUSPEND|RESUME} MP [messasge
publisher code]
```

```
./sess-control.sh {QUERY|START|STOP|SUSPEND|RESUME} MS [messasge sender
code] [RESUME_SUB]
```

```
./sess-control.sh {QUERY|SUSPEND|RESUME} SA [source application code]
```

```
./sess-control.sh {QUERY|RESUME} SB [subscriber code]
```

```
./sess-control.sh {START|STOP} EM
```

Note Use this to start and stop the e-mail service.

```
./sess-control.sh {START|STOP} AR
```

Note Use this to start and stop the archive service.

Where `{QUERY|START|STOP|SUSPEND|RESUME}` is the command you want to execute.

Example To start all services, enter the following:

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

To stop the archive service, enter the following:

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

To suspend an event service named ES1, enter the following:

```
./sess-control.sh SUSPEND ES ES1
```

To resume a message publisher service named MP1, enter the following:

```
./sess-control.sh RESUME MP MP1
```

Note If the script is going to be called from a cron job, the line tail `-f $QXODIR/logs/$1.log` must be removed from the end of the script.

32-bit Startup Option

In the QXO `start-sess.sh` file, a new parameter can be appended to the `-param` entries to indicate that QXO will process rowids as 32-bit. This optional parameter looks like this:

```
-param "$1",$LOGLEVEL,yes
-param "$1",$LOGLEVEL,no
```

The default is no, so it can be omitted.

QXO runs on version 10.1B, which processes rowids as 64-bit; earlier versions of Progress send rowids as 32-bit.

By default, QXO converts all incoming 32-bit rowids to 64-bit. However, users that are upgrading from QXO versions prior to version 1.4 must run a conversion routine to convert all rowids stored in the database—that is, the link to the business object message—to 64-bits.

Because this conversion can take a long time to complete—especially if the QXO system has been operational for a long time—this parameter setting is provided as a workaround until a suitable time is found to perform the conversion and/or archive the old data.

This is a system-wide setting, so if an QAD Enterprise Applications system was using 64-bit rowids and QXO was processing them as 32-bit, erroneous results may arise.

Chapter 9

Using the QXtend Message Monitor

The following material covers the QXtend Message Monitor.

Note The QXtend Message Monitor is available only in the .NET user interface.

QXtend Message Monitor 118
Explains how to use the QXtend Message Monitor with details on the profile message summary, Subscriber Messages tab, Subscriber Responses tab, and Subscriber Message Details.

QXtend Message Monitor

Profile messages are generated by QXO and delivered to external subscribers. The QXtend Message Monitor allows you to track these outbound profile messages, as well as the response messages that are returned from subscribers. You can view the complete lifecycle of profile messages through QXtend—from publication and delivery through to response.

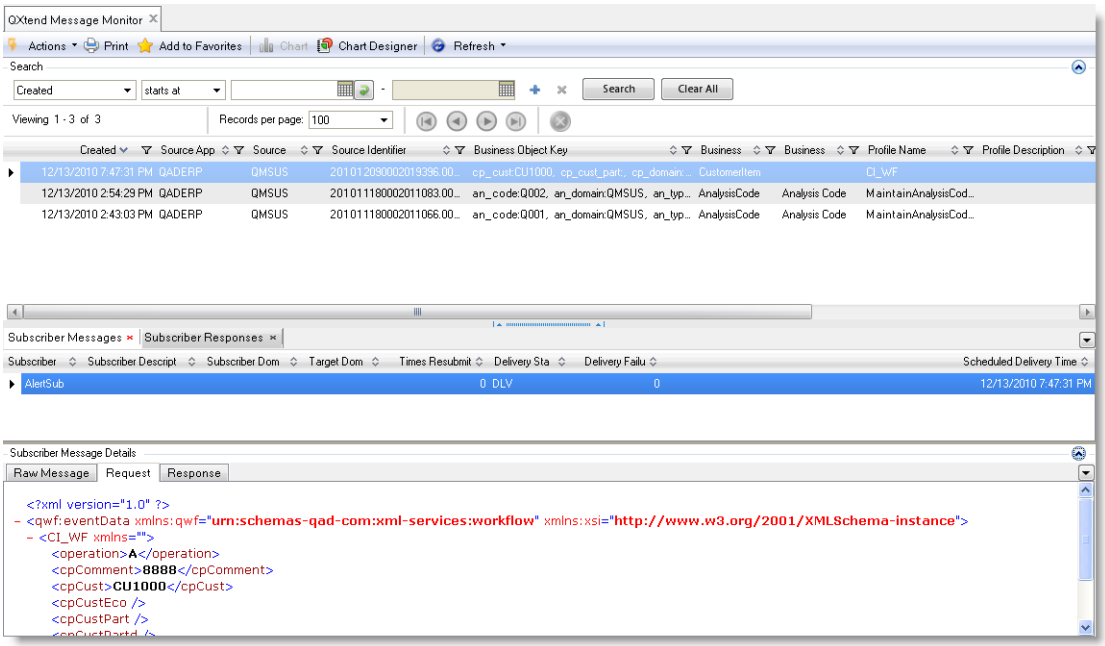
A profile message is created for each profile that is registered against a business object. Response messages are generated by the destination application and returned to QXtend Outbound.

You can use the QXtend Message Monitor to view:

- A summary of profile messages delivered to subscribers
- Details of subscriber messages, including delivery status
- Details of subscriber responses arising from message processing
- Raw XML data for messages, requests, and responses

The information displayed in the tables can be filtered using various criteria to make it easier to understand the data displayed. Profile messages can be filtered by date and time; subscriber, source application, source domain, target domain, or unique identifier (OID or rowid); and delivery status. Profile messages can also be grouped by profile type. For details of filtering and grouping data in tables, see the section “General UI Functions” on page 19.

Fig. 9.1
QXtend Message Monitor



The Outbound Message Monitor contains the following:

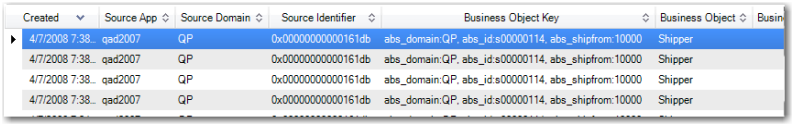
- Profile message summary
- Subscriber Messages tab
- Subscriber Responses tab
- Subscriber Message Details area

Profile Message Summary

The profile message summary, located at the top of the QXtend Message Monitor screen, describes profile messages that have been delivered to subscribers. Clicking a profile message causes the associated subscriber messages and subscriber responses to display in the relevant tabs.

A single profile message may have one or multiple related subscriber messages. If a subscriber message is delivered successfully on first submission, only one subscriber response exists for that message. If delivery is unsuccessful, a message may have multiple subscriber responses describing the processing errors.

Fig. 9.2
Profile Message Summary



Source Domain. The domain in the source application that was the source for the event message.

Source Rowid. The unique identifier (OID or rowid) of the application database record updated by the event.

Business Object Key. The fields selected as primary in the definition of the QXtend business object.

Business Object. The QXtend business object associated with the event message.

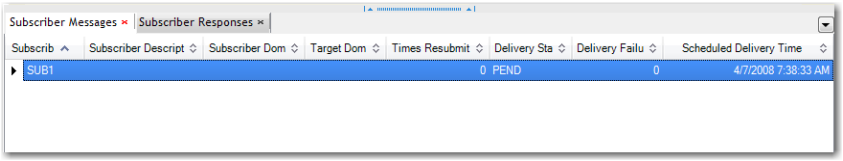
Profile Type. The application type of the subscriber for which the profile message is intended. For details on profile types, see “Modifying Profiles” on page 89.

Subscriber Messages Tab

The Subscriber Messages tab shows response messages received from the subscriber. Response messages are only received from subscribers if a Web service is used.

The mpt_msg_subs_resp table in the QXtend database stores the response XML and processing result. The mpt_msg_subs_excp table stores any exceptions raised on the related response message.

Fig. 9.3
Subscriber Messages Tab



Subscriber Domain. Specifies the domain in the source application from which the subscriber solely wants to receive event messages. Event messages originating from other domains are ignored.

Delivery Status. The delivery status of the profile message as reported by the receiving application.

APPERR. Destination Application Error. An application error has occurred in the destination application (QAD Enterprise Applications, for example).

DLV. Message Delivered Successfully to Subscriber. Response should show success for a DataSync profile type as it was successfully delivered to a Web service subscriber.

For a subscriber using file directory service delivery the response will be:

```
<QdocResponse>No Response XML Available</QdocResponse>
```

because no parsing of the response has been performed. The DLV status here means the message was successfully written to the target directory on the file system.

HOLD. The message not been sent and is being held up by a previous message (for the same rowid) that has an error status. The message that is in error needs to be corrected before the next message can be sent.

PEND. Pending Message. No response has been generated yet as the message is waiting to be sent.

SOAPERR. SOAP Fault Generated by QXI. Generated by the QXI Web service. The most common SOAPERR is:

```
<number>QdocExceptionex005</number>
<description>QDoc is not supported</description>
<severity>error</severity>
```

SNDERR. Infrastructure Errors. An infrastructure error exists in the subscriber. This error may occur if there was an internal server error or no response was received from the Web service, indicating there was a problem communicating with the service.

SUP. Subscriber Message Not Sent, Superseded by Later Profile Message. No response is generated because the message was superseded by a later message.

WRN. Warning Exceptions Received. A warning has been returned from the destination application. For example:

```
A record was in use by another user: aud_det in use by mfg,25 on pali505 3. Wait or press
CTRL-C to stop. (121)
```

Delivery Failures. The number of times the profile message failed to be delivered successfully to the subscriber.

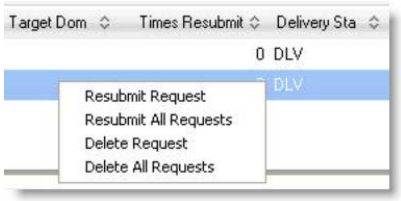
Resubmitting or Deleting Requests

Right-click a subscriber message in the Subscriber Messages tab to resubmit or delete the QDoc. You can choose:

- Resubmit Request to resubmit the selected subscriber message.
- Resubmit All Requests to resubmit the profile message to all related subscribers
- Delete Request to delete the selected subscriber message.
- Delete All Requests to delete all subscriber messages.

Note Messages are resubmitted according to the sending option defined for the subscriber. Messages for subscribers using the Send Immediately option are resubmitted immediately. Messages for subscribers using the Use Delivery Schedule option are resubmitted at the next scheduled delivery time.

Fig. 9.4
Resubmit Menu



You can only resubmit a QDoc that has previously been delivered. If a superseded message is selected for resubmission, the latest version of the message is resubmitted to the subscriber. You cannot resubmit a pending message.

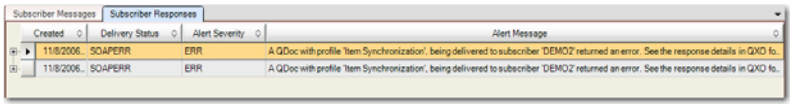
During resubmission the profile message and subscriber detail records are locked to prevent another process from trying to resubmit the same QDoc message.

An alert indicates whether message resubmission was successful or unsuccessful.

Subscriber Responses Tab

The Subscriber Responses tab shows message responses returned from the subscriber. A message that is delivered successfully will have only one response. A message that has resubmitted can have potentially multiple subscriber responses indicating why delivery failed. You can expand subscriber responses for messages that have multiple errors.

Fig. 9.5
Subscriber Responses Tab



Alert Severity. The severity of the exception. Possible values are ERR, WRN, or MES (indicating successful delivery).

Alert Message. Descriptive text for the alert message. This alert text cannot be changed.

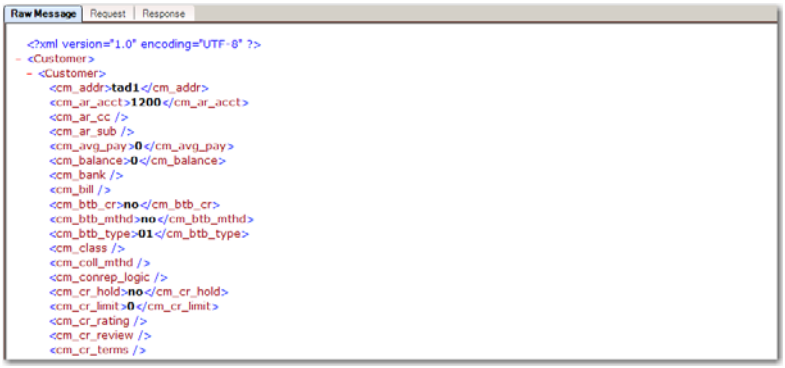
Number. This column displays only when a subscriber response is expanded. Contains the QXI exception code related to the error. For details about exception codes, see page 349.

Context. This column displays only when a subscriber response is expanded. Indicates the frame or field within QAD Enterprise Applications where QXtend Inbound failed during message processing.

Subscriber Message Details

The Subscriber Message Details tab displays an XML code representation of the raw message event, subscriber message, or subscriber response. Here the Subscriber Message Details tab displays the raw message XML of a profile message request.

Fig. 9.6
Subscriber Message Details



Click a tab to view the XML code. The XML is retrieved from the database when a message detail tab is selected. You can also expand and contract the Subscriber Message Details tab as required.

Raw Message. Displays the latest version of the data that has been extracted for a specific business object, represented as an event message. Only the current version of this information is stored in the QXO database.

Request. The profile message delivered to the subscriber.

Response. The data sent in response to a profile message. No XML response data ia available for subscribers that use the file directory service method of delivery.

Chapter 10

QXtend Query Service

This chapter describes the Query Service in QXtend Outbound, which enables calling applications to access data within a QAD application, without replicating the data.

Introduction

The Query Service is a QXO service that provides a data interface between QAD products, and between QAD products and external systems, without the need to connect to an application database.

The Query Service is a service interface API and can be called using a native Progress call, a Progress appserver call, or an XML Web service (QXI) call. When called, the Query Service connects to the source application database, extracts the required data, packages the data, and returns it to the calling application. The data can be returned as either XML or as a ProDataSet.

The Query Service enables the calling application to access data within a QAD application without replicating the data to the calling application. For example, the calling application can use the Query Service to access country codes stored in QAD EE without connecting directly to the database or copying the data. The connection is fast and in real time, and the calling application does not require knowledge of the source. The QXO event service and event publisher processes access the data, without updating the QXO database.

QXI is not required for the Query Service unless you want to issue XML requests. When you make an XML Web service call to the Query Service, QXI receives the request and calls the Query Service in QXO to process it.

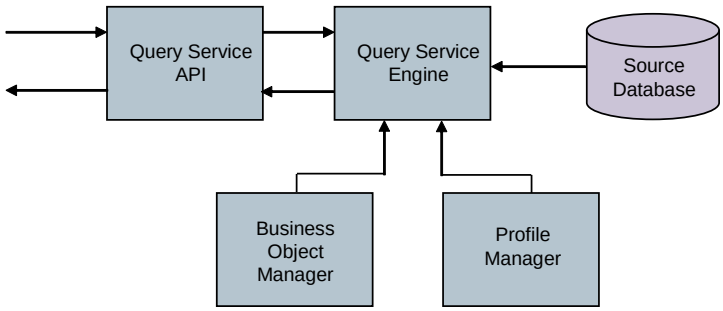
You can use the Query Service to:

- Access data in other QAD applications for lookups and validation. This facility removes the need to replicate data into the product module.
- Allow an external application to access data stored in a QAD application. This facility could be used to validate data from an external application before it is sent to the QAD application, reducing potential errors when interfacing between products.

Query Service API

The Query Service is an SI API that is called by using a native Progress call, a Progress appserver call, or an XML Web service (QXI) call. The API is serviced by the Query Service engine, Business Object Manager, and Profile Manager.

Fig. 10.1
Query Service API



The Business Object Manager and Profile Manager create the business object and profile datasets. The Query Service engine connects to the source application database, extracts the data, and returns it to the calling application as a profile dataset.

Note If a business object is domain-enabled, the domain must be passed in the request. The domain code is a session context parameter and does not have to be included in the query filter.

When accessing the API, the calling application can specify filter conditions that control the data returned by the Query Service. These are defined as a single-character expression or string, and the data can only be filtered using the primary table in the business object. When using XML (QXI) to call the Query Service, the filter is a node in the request XML. When using a direct API call, a field in a temporary table is passed as a parameter.

Use a native Progress program called `com/qad/qxtend/si/QueryService.p` to access the Query Service API. The internal procedure is called `querySourceApplication`. It takes a static dataset as an input parameter and a dataset-handle as an output parameter.

The input dataset contains the definition of the Query Service request. The temporary table in the data set is defined as follows:

Table 10.1
Input Dataset for a Query Service Request

Property Name	Description
SourceApplication	The name of the source application defined in QXtend Outbound. It contains the connection information for the database that is the source of the query data.
Profile	The name of the profile (or business object) defined in QXtend Outbound that defines the format of the response.
Filter	The filter to apply to the profile (or business object) to return a subset of data from the database. This must be a valid Progress ABL WHERE clause.
MaxRows	The maximum number of records to return. If set to zero, the Query Service will return all data that matches the filter.
IgnoreBOFilter	If set to TRUE, only the filter provided in the Query Service request will be used. If set to FALSE, the filter defined on the business object is added to the filter defined in the Query Service request.
IgnoreInnerJoin	If set to TRUE, any inner join contained in the business object is ignored.

The API first validates the input data, and then calls the Query Service engine to process the request.

Query Service Setup

The Query Service provides the following methods for calling the API:

- Direct connection
- Progress appserver connection
- Web service

Direct Connection Setup

When calling the Query Service using a direct connection, you must connect the Progress session to the QXO database. In addition, ensure that the QXO runtime directory is included in the PROPATH.

Example

```
/* Query Service Direct Connection Example */
define variable goQueryServiceEngine as class com.qad.qxtend.qxo.QueryServiceEngine.
define variable hResponse as handle no-undo.
{com/qad/qxtend/qxo/dsQueryServiceRequest.i}
{com/qad/qxtend/qxo/ttException.i}

goQueryServiceEngine = new com.qad.qxtend.qxo.QueryServiceEngine().

create QueryServiceRequest.
assign
   QueryServiceRequest.SourceApplication = "ERP32"
   QueryServiceRequest.Profile = "Customer"
   QueryServiceRequest.Filter = "cm_addr >= '0100' and cm_addr <= '100'"
   QueryServiceRequest.MaxRows = 2.

goQueryServiceEngine:querySourceApplication
    (input dataset dsQueryServiceRequest by-reference,
    input "QP", /* Domain */
    output dataset-handle hResponse).

if goQueryServiceEngine:hasExceptions() then do:
   goQueryServiceEngine:getAllExceptions(output table ttException).
   for each ttException:
      message ttException.cDescription view-as alert-box.
   end.
end.
else do:
   hResponse:write-xml("file","qs.xml",true).
end.

delete object goQueryServiceEngine.
```

In the example, QXtend is connecting to a source application called ERP32, the profile is called Customer, a filter is defined, and the requests can return a maximum of two records. The result of the Query Service is returned in the dataset-handle, `hResponse`. The dataset is then serialized to the file `qs.xml`.

If you want to analyze the data extracted, parse the dataset dynamically.

Progress AppServer Setup

Configuring the AppServer

For details on configuring the AppServer see “Configuring the AppServer” on page 140.

Defining the Service Interface Code

To call the Query Service using the Progress appserver, you must define the Service Interface code in the PROPATH, as shown here:

Example

```
/* Query Service AppServer Connection Example */

define variable hResponse  as handle no-undo.
```

```
define variable hServer    as handle no-undo.
define variable lConnected as logical no-undo.

{com/qad/qxtend/qxo/dsQueryServiceRequest.i}
{dscontxt.i}
{dsexcptn.i}
{com/qad/qra/si/dsNull.i}

create server hServer.
lConnected = hServer:connect("-AppService qxosi_AS -H nately -S 5162","","").
if not lConnected then do:
   message "Error connecting to AppServer" view-as alert-box.
   return.
end.

run createContext(input "QAD",
                  input "programName",
                  input "com/qad/qxtend/si/QueryService.p").

run createContext(input "QAD",
                  input "domain",
                  input "QP").

run createContext(input "QAD",
                  input "methodName",
                  input "querySourceApplication").

create QueryServiceRequest.
assign
    QueryServiceRequest.SourceApplication = "qad2007"
    QueryServiceRequest.Profile = "Customer"
    QueryServiceRequest.Filter = "cm_addr >= '0100' and cm_addr <= '100'"
    QueryServiceRequest.MaxRows = 2.

run com/qad/qra/si/RPCRequestService.p on server hServer
    (input-output dataset dsSessionContext,
    output dataset dsExceptions,
    input dataset dsQueryServiceRequest by-reference,
    input-output dataset dsNullInOut,
    output dataset-handle hResponse).

if can-find(first temp_err_msg) then do:
   for each temp_err_msg:
message temp_err_msg.tt_msg_desc view-as alert-box.
   end.
end.
else do:
   hResponse:write-xml("file","qs.xml",true).
end.

hServer:disconnect().
delete object hServer.

procedure createContext:
   define input parameter propQual as character no-undo.
   define input parameter propName as character no-undo.
   define input parameter propVal  as character no-undo.

   create ttContext.
   assign ttContext.propertyQualifier = propQual
          ttContext.propertyName      = propName
          ttContext.propertyValue     = propVal.

end procedure.
```

The system populates the Context temporary table with the values required for the Service Interface.

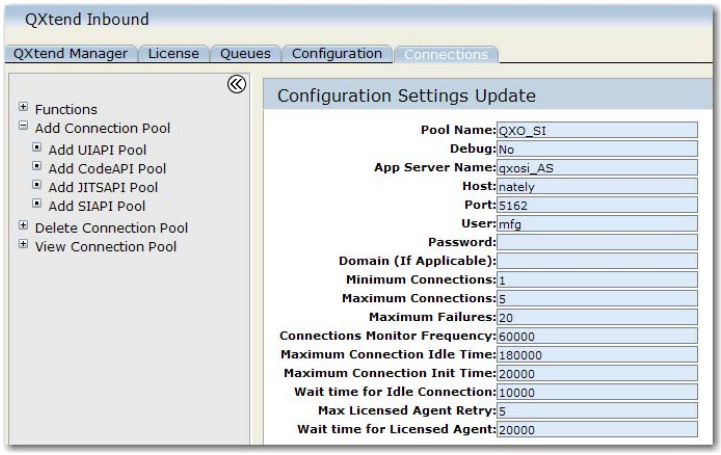
Web Service Setup

Connection Pool

To call the Query Service Web service using QXI, you must create an adapter to connect to the Progress appserver.

Create the adapter by configuring a connection pool for an SI API, as shown in Figure 10.2. See “QXI Connection Pool Manager” on page 201 for details on connection pools.

Fig. 10.2
Configuration Settings



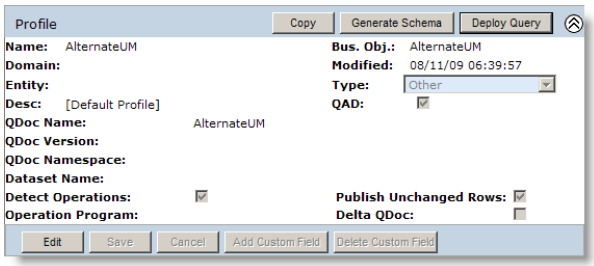
You must also create a corresponding receiver for the connection pool. See “Inbound Receivers” on page 180 for details on receivers.

Deploying the Query API

You publish the Query Service API for the selected profile to a receiver in QXI by using the Deploy Query button in QXtend Outbound. After establishing a connection to the QXI server, you select a receiver, deploy the API, and then verify the deployment has been successful.

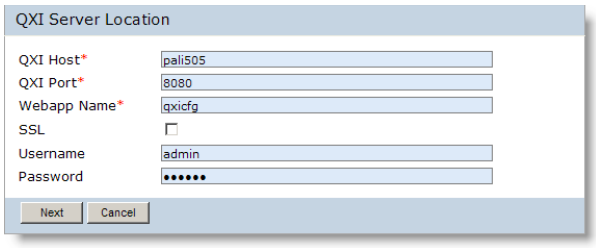
1. In QXtend Outbound select the Profiles tab.
2. Select the profile you want from the Profiles node in the application menu.
3. Click the Deploy Query button located at the top-right corner of the Profile window.

Fig. 10.3
Deploy Query Button



4 The QXI Server Location window displays.

Fig. 10.4
QXI Server Location



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.

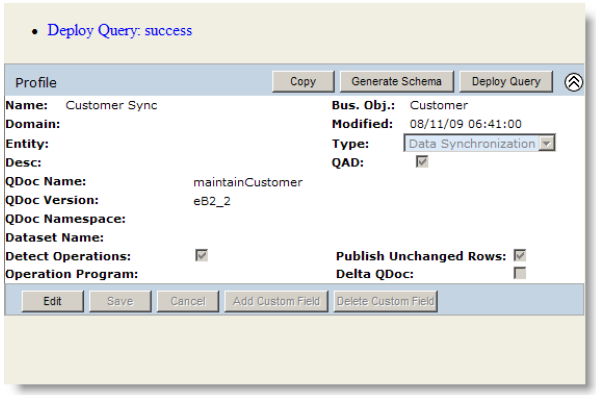
Note The system assumes the same user login credentials as for QXtend Outbound.

The Receivers screen displays 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 You are returned to the Profiles screen. A system message displays the status of the deployment.

Fig. 10.5
Deployment Status Message



Chapter 11

QXtend Interaction with Financials

This chapter describes configuration required for QXtend to receive business object event data from QAD EE Financials.

Introduction

QXtend can accept and publish raw business object event data from the QAD EE Financials module, eliminating the need to extract data from the Financials database. Financials can use QXtend to:

- Replicate financial data, including configuration and transactional data, between separate instances of Financials.
- Replicate financial data between Financials and a third-party financial application.

Financials business objects employ DDP and are passed directly from the source to the QXODB database in QXO. From the QXODB database they are processed by the message publisher.

The Financials raw business object event data is saved in an XML file and passed directly to QXO using an event message API; the resulting QDoc is published without the need for any further processing.

Financials posts data to QXtend using event publishing—a form of DDP—which lets you publish a message when a particular business event occurs within Financials. The message is sent to a messaging bus, which then routes the message to a specific destination.

Event publishing is typically used to notify an external resource that a certain condition has occurred within Financials. For reasons of dependency, transaction integrity, and speed, the Financials application has a buffer queue that is used within the running transaction to prepare messages before they are sent out. This also ensures that no events are lost during system down-time and that no disruption of service occurs.

Setup is required in both QXtend and Financials to enable data to be replicated between the two applications.

Financials to QXtend Outbound Setup

Financials uses Event publishing to notify QXtend that it has published business objects that must be replicated. Event publishing has three components:

- Event Destination

  Use Event Destination to define the name of the event and the type of messages it will contain. This points to the Progress AppServer instance for QXtend.

- Event Configuration

  Use Event Configuration to activate the publishing mechanism, configure which components and status transitions will trigger events (for example, supplier invoice creation), and indicate the destination queue of the event.

- Event Daemon

  The Event daemon reads records in its queue and sends them to the corresponding broker.

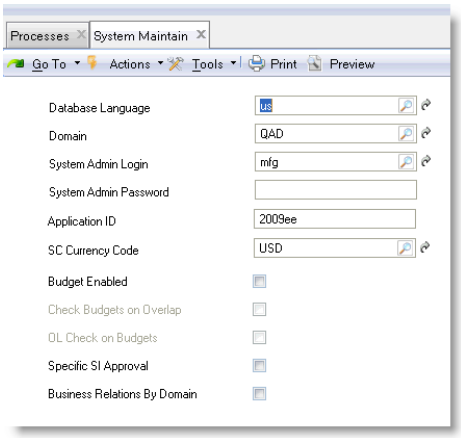
Note See *User Guide: QAD System Administration* for details on the event daemon and event publishing.

Financials Event Setup

This section describes how to configure Financials to publish events to QAD QXtend.

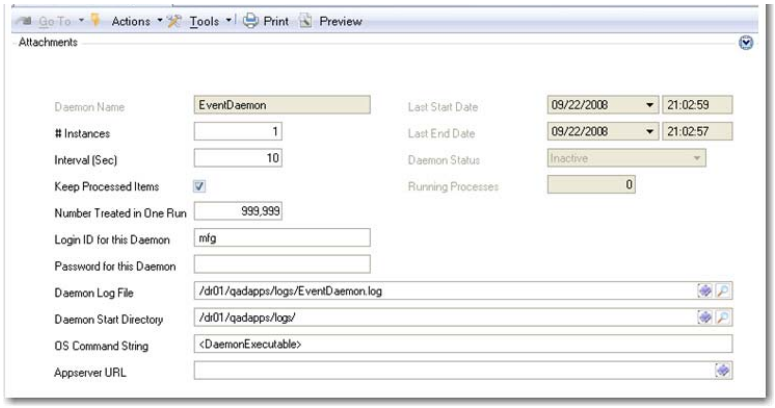
1 Set the Application ID in System Maintain (36.24.3.1). The application ID value should be the same as the source application name configured in QXO.

Fig. 11.1
System Maintain (36.24.3.1)



2 Configure the Event Daemon using Event Daemon Configure (36.14.16.16.1). Otherwise, the Event Daemon cannot be started.

Fig. 11.2
Event Daemon Configure

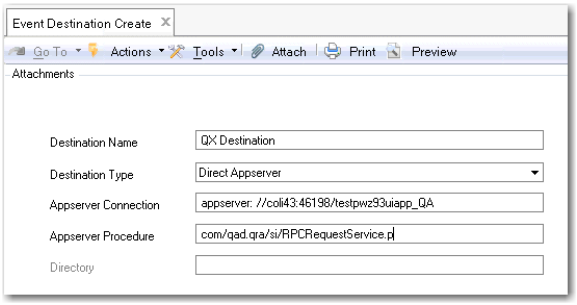


3 Record the location of the QXtend appserver in Event Destination Create (36.14.16.14.1).

Note For details on configuring the AppServer see Chapter 12, “Setting Up the Service Interface AppServer,” on page 139.

4 In the AppServer Procedure field of Event Destination Create, specify the `RPCRequestService.p` procedure. This is the general service interface dispatcher.

Fig. 11.3
Event Destination Create (36.14.16.14.1)

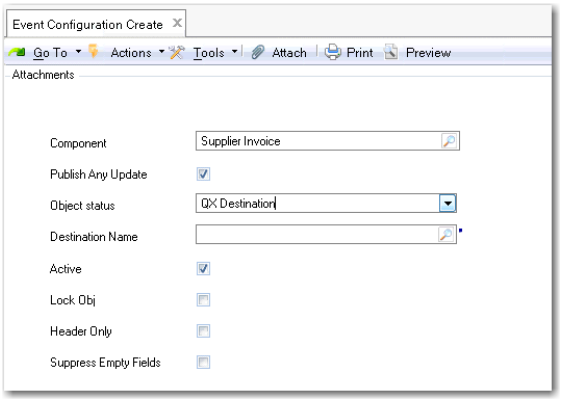


5 In Event Configuration Create (36.14.16.15.1), select the component and then select the Publish Any Update check box.

Note If the workspace has changed, an Event Configuration record must be created for the new workspace.

6 Specify the destination previously created in the Destination field.

Fig. 11.4
Event Configuration Create (36.14.16.15.1)

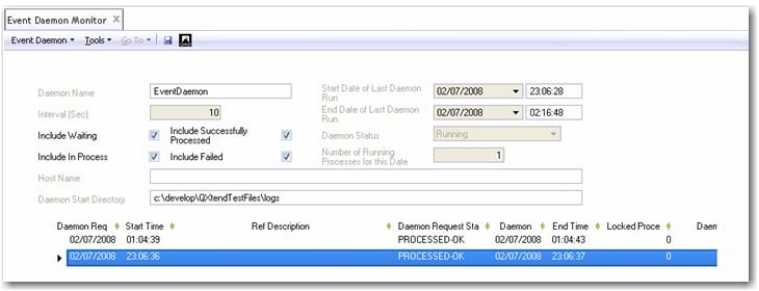


7 Start the Event daemon using Event Daemon Start (36.14.16.16.4).

8 Change a record on an existing supplier invoice using Supplier Invoice Modify in Financials.

9 Open the Event Daemon Monitor to view the event generated in Financials when you modified the supplier invoice.

Fig. 11.5
Event Daemon Monitor (36.14.16.16.2)



10 Open Supplier Delete (28.1.1.9) and delete a supplier record.

11 Open Event Daemon Monitor.

Create, modify, and delete activities can generate a published event from Financials.

Configuring the Connection Pool and Schema (QXI)

Note You configure the connection pool and schema in QXI.

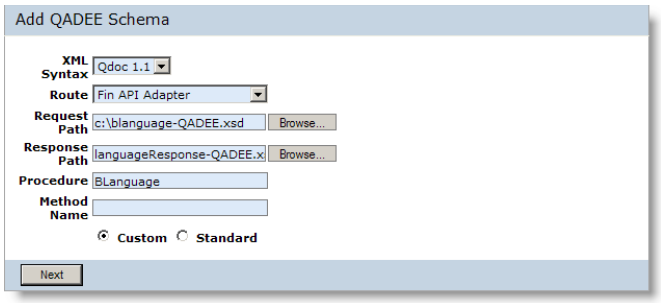
1 Create a connection pool for the FIN API adapter that connects to the QXtend appserver. See Chapter 17, “QXI Connection Pool Manager,” for details.

2 Load the XML schema. The XML schemas are stored in the location specified in the following directory on the appserver:

`c:\<installdir>\qadfin\components\progress\xml\*.xsd`.

a Rename the file with the QDoc version appended to the file name.

Fig. 11.6
Adding the Schema



b Enter the following values for the schema:

XML Syntax: QDoc 1.1.

Route: FIN API Adapter.

Request Path: The location of the schema file from Financials.

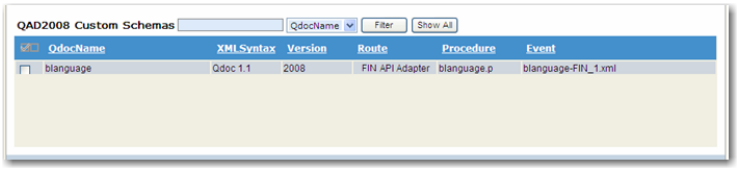
Response Path: Blank, or specify a response schema.

Procedure: Enter the name of the Financials component; for example, BLanguage.

See “QDoc Schemas” on page 187 for details on adding schemas.

3 Register the API with the receiver. See “Adding a Schema to an Existing Receiver” on page 183 for details.

Fig. 11.7
Custom Schema

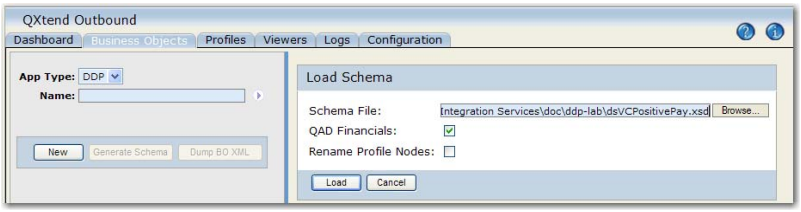


Direct Data Publish Setup in QXO

1 On the Business Objects tab, select the source application type you want to create a new DDP business object for. Click New and then click Yes on the New Business Object pop-up box to create a DDP business object.

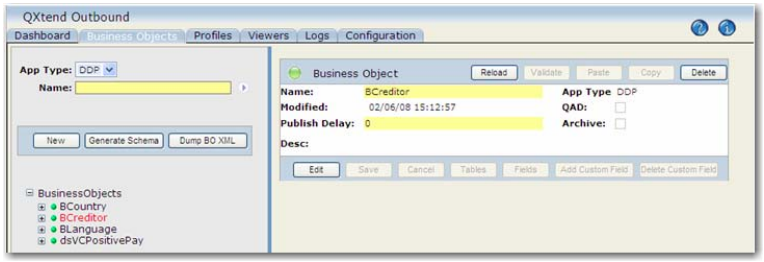
2 Create a business object by loading the Supplier Invoice schema.

Fig. 11.8
Load Schema



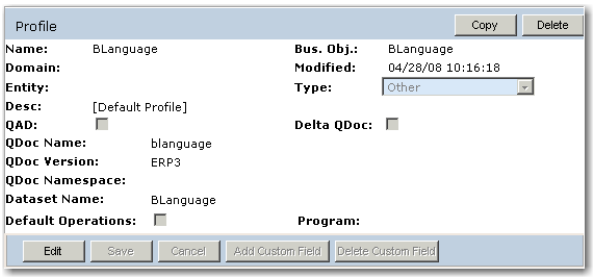
The schema is loaded.

Fig. 11.9
Loaded Schema



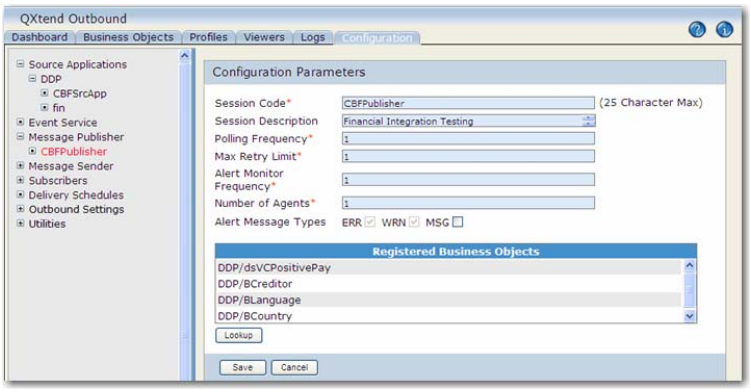
QXtend creates the default profile automatically.

Fig. 11.10
Default Profile



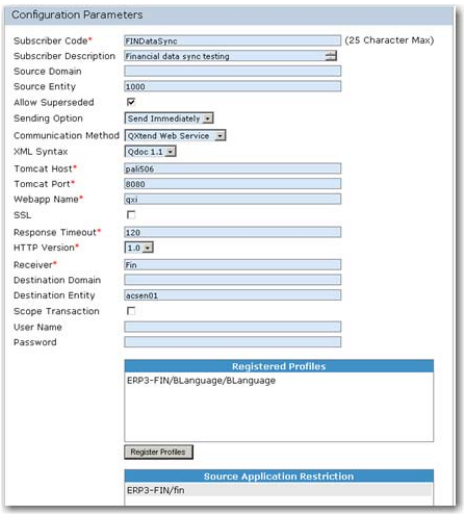
3 Create a Message Publisher for DDP.

Fig. 11.11
Message Publisher



4 In the Subscribers Configuration Parameters screen, create a message subscriber for DDP.

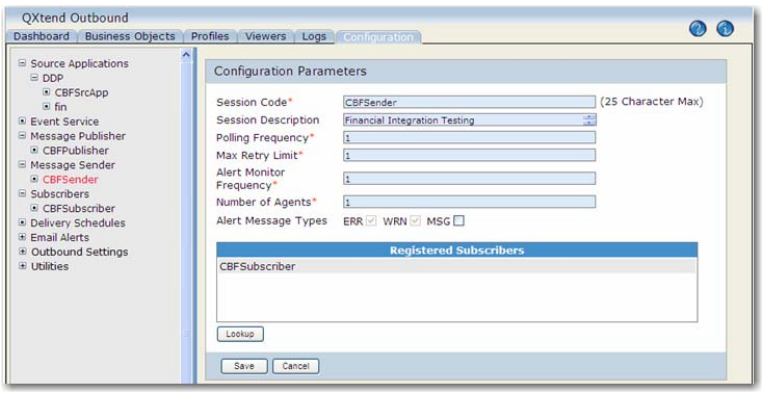
Fig. 11.12
Subscribers Configuration Parameters



Note Register the profile and source application created.

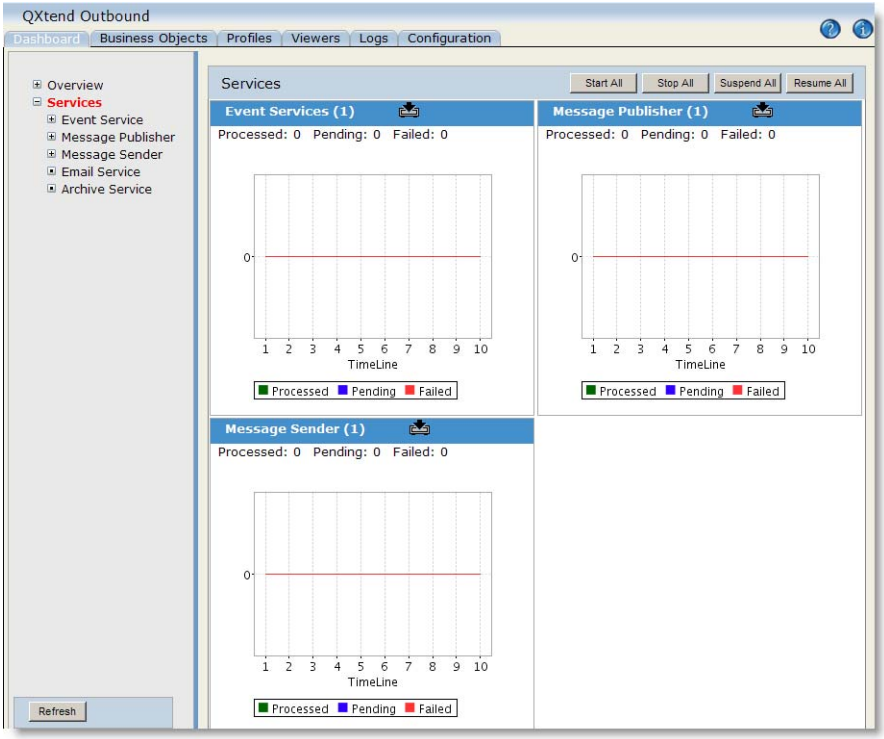
5 In the Message Sender screen of the Configuration tab, create a sender.

Fig. 11.13
Message Sender Configuration Parameters



6 In the Services screen in the Dashboard tab, click the Stop All button and then click the Start All button.

Fig. 11.14
Services



Chapter 12

Setting Up the Service Interface AppServer

This chapter describes how to configure the service interface AppServer, which is used for both the Query Service and direct data publish.

Introduction

The AppServer is used for both the Query Service and direct data publish. The service interface AppServer is configured during the installation of QAD QXtend. By default the name of the AppServer is `qxosi_AS`, but this name may have been changed during installation; check the `ubroker.properties` entry for the AppServer.

Note If the AppServer has not been configured, see “Configuring the AppServer” for details.

Configuring the AppServer

To configure the AppServer you can use two methods—the result is the same.

- Directly edit the `ubroker.properties` entry for the AppServer.
- Use dialog boxes to populate the `ubroker.properties` entry with the correct values.

ubroker.properties

You can configure the AppServer by editing the `ubroker.properties` entry. The following is an example of the `ubroker.properties` entry required for the AppServer:

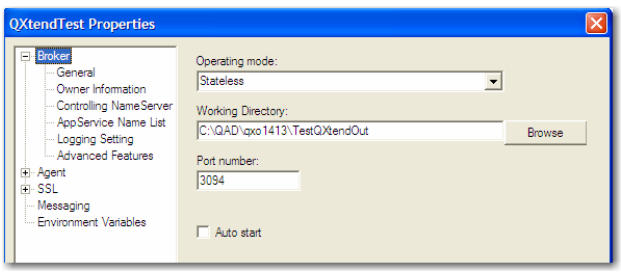
```
[UBroker.AS.qxosi_AS]
    appserviceNameList=qxosi_AS
    autoStart=1
    brokerLogFile=<QXO Install Dir>/logs/qxosi_AS.broker.log
    controllingNameServer=NS1
    initialSrvrInstance=1
    maxSrvrInstance=5
    minSrvrInstance=1
    operatingMode=Stateless
    portNumber=10001
    PROPATH=<QXO Install Dir>/runtime:<QXO Install Dir>
    registerNameServer=1
    registrationMode=Register-IP
    srvrLogFile=<QXO Install Dir>/logs/qxosi_AS.server.log
    srvrStartupParam=-db qxodb -S qxodb-service -ld qxodb -H host
    srvrStartupProc=com/qad/qxtend/si/AppServerStart.r
    uuid=11d1def534ea1be0:e48e1b:11833ce3369:-8000
    workDir=<QXO Install Dir>
```

Configuration Dialog Boxes

Follow these instructions to configure the AppServer by using configuration dialog boxes.

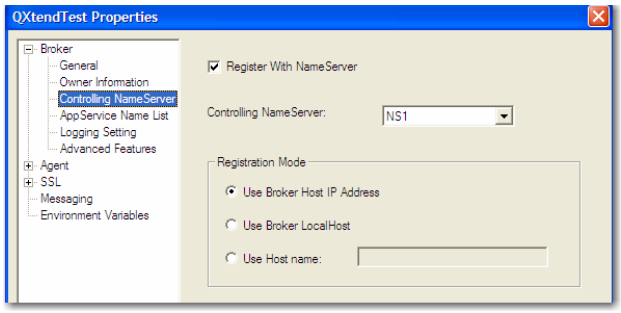
1 In the properties for the Direct Data Publish AppServer, record the path to the working directory for QXtend Outbound.

Fig. 12.1
Broker Properties



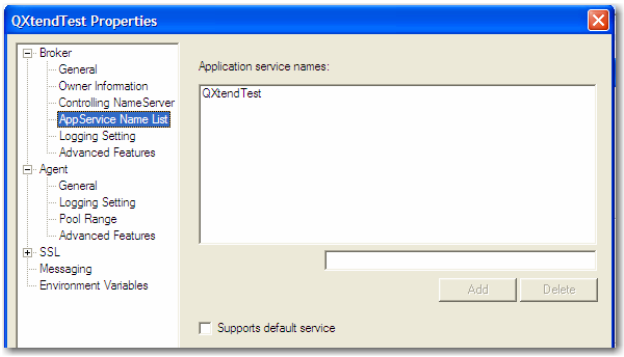
2 Verify the NameServer with which the AppServer registers. This normally defaults to NS1.

Fig. 12.2
Controlling NameServer Properties



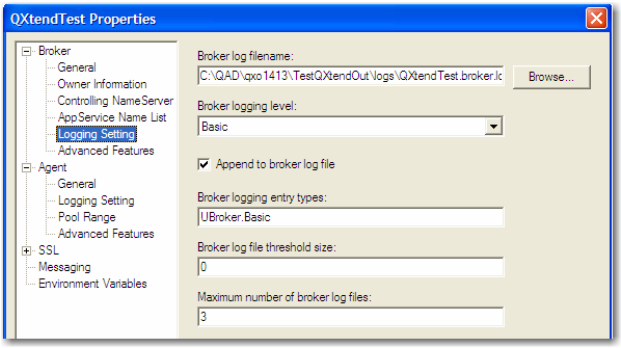
3 Verify the name of the DDP AppServer.

Fig. 12.3
Broker AppService Name List Properties



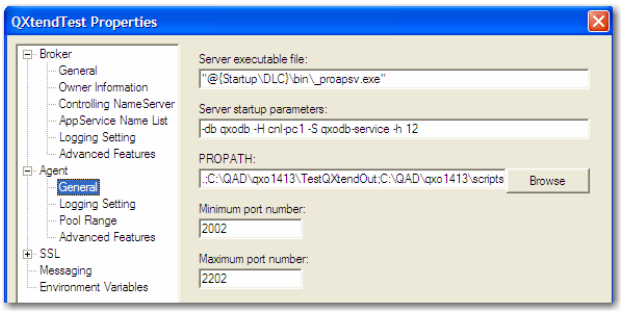
4 In the Broker Logging properties, specify the broker log filename and set the broker logging settings as required. The default settings may be suitable.

Fig. 12.4
Broker Logging Setting Properties



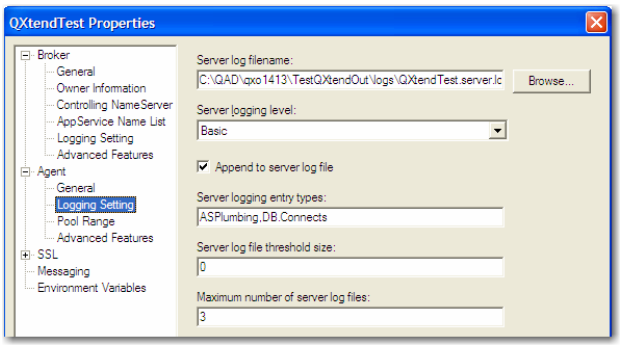
5 In Agent General Properties, define the PROPATH of the DDP AppServer.

Fig. 12.5
Agent General Properties



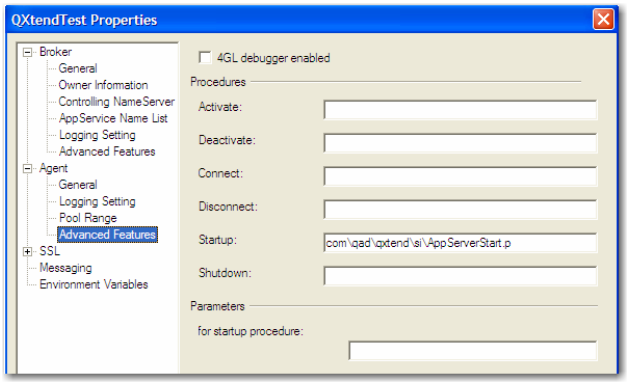
6 In the Agent Logging Setting properties, specify the agent log filename and set the agent logging settings as required. The default settings may be suitable.

Fig. 12.6
Agent Logging Setting Properties



7 In the Agent Advanced Features properties, define the startup program for the DDP AppServer.

Fig. 12.7
Agent Advanced Features Properties



Section 2

QAD QXtend Inbound

This section contains information on implementing and using QAD QXtend Inbound (QXI).

Chapter 13

QXtend Inbound Overview

This section provides an overview of the QXtend Inbound (QXI) environment.

QXtend Inbound Overview

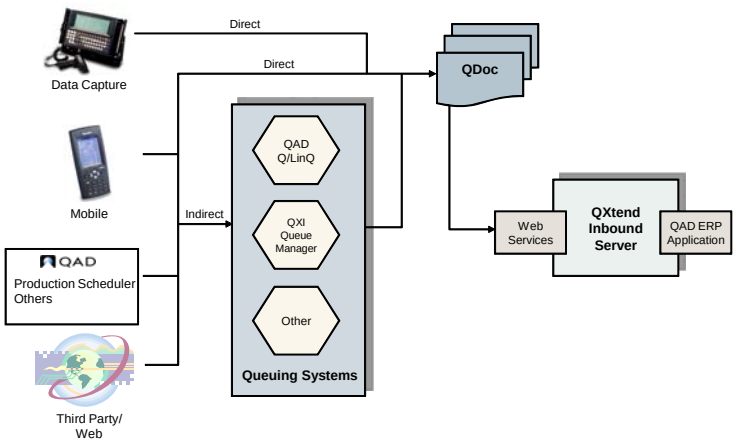
QXI is a component of the QAD QXtend interoperability framework. This framework provides a standardized data interface between QAD products, and between QAD products and external systems. The interface is a Web services-based, SOAP-compliant, XML framework, enabling complete platform-independent access to QAD Enterprise Applications business functionality. See “Glossary” on page 415 for definitions of terms and abbreviations.

The framework consists of:

- QXI, which imports SOAP-compliant XML QDocs from external applications into QAD Enterprise Applications and other QAD applications such as QAD Production Scheduler.
- QAD QXtend Outbound (QXO), which exports user-configurable business objects as XML QDocs out of QAD products such as QAD Enterprise Applications.

QXI delivers inbound data to menu-level programs in QAD Enterprise Applications eB and above. The inbound data is in the form of XML data documents, called QDocs, delivered in a proprietary XML format. QXI also has security mechanisms in place to ensure that only authorized personnel can update data to menu-level programs. Prior to all data processing, the system checks whether the user has access rights to corresponding menus in QAD Enterprise Applications. If the user does not have access rights to the menu, an error is displayed. Figure 13.1 illustrates the QXI system architecture.

Fig. 13.1
Basic QXI System Architecture



As shown in Figure 13.1, incoming documents—called requests—can be generated directly from data capture tools such as barcode readers; mobile devices; other QAD products, QAD Production Scheduler, or QAD EDI ECommerce; and from any third-party application that can generate XML.

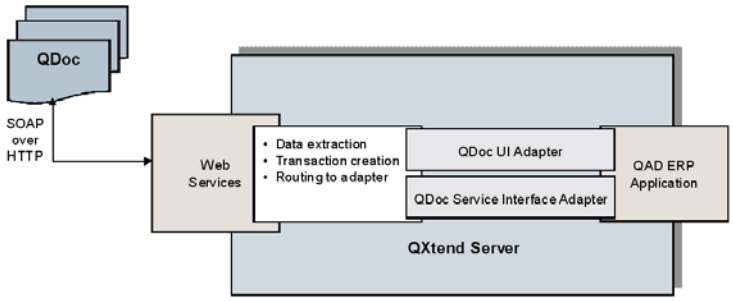
The data is formatted as SOAP-compliant XML documents using QDoc templates in XML schema format. These documents are fed directly to QXI. Alternatively, you can queue the documents using a number of queue management interfaces, including the QXI Queue Manager and the QAD Q/LinQ product. See Chapter 21, “QXtend Inbound with QAD Q/LinQ,” on page 245 for more details.

In addition, you can queue XML documents that do not meet QXI standards, and call the QXI transformation engine to apply custom XSLT style sheet transformations to the document. The transformation engine outputs a valid QDoc request. See “QXI Transformation Engine” on page 166 for more details.

QXI accepts the XML QDoc requests as Web service calls as shown in Figure 13.2. Optionally, you can send the incoming QDocs through the QXI preprocessor to validate the incoming data, eliminating incorrect QDocs. You can also create postprocessing programs to validate response data. See Chapter 19, “QXtend Inbound Pre- and Postprocessors,” on page 233 for details.

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

Fig. 13.2
Server Internals



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

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

Direct API Programs

The direct service interface APIs enable the transfer of data between one instance of the QAD Enterprise Applications Financials module and another instance, or between Financials and external systems.

The inbound data format for service interface APIs is XML documents in QAD-proprietary format.

Note To use service interface APIs you must set up a Progress AppServer. See Chapter 20, “Configuring the Progress AppServer,” on page 239 for additional information on AppServer setup.

QDocs

Each inbound XML QDoc request results in an XML QDoc response. Each QDoc references the QAD Enterprise Applications session and version and the target QAD Enterprise Applications application program. A QDoc can contain one or more records for that calling program, but cannot call other programs in QAD Enterprise Applications. See Chapter 23, “QDoc Structure Reference,” on page 265 for examples of QDoc files.

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

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

QDoc XML Schemas

All standard XML schemas are available on the Web by clicking the QDoc link under Downloads on the QAD Support Web site at:

http://support.qad.com/

Note Depending on the QDoc syntax specification being used, the way in which schemas are used by QDocs differs. Refer to the appropriate section for the relevant QDoc schema syntax specification.

QDoc XML Schemas (1.1)

Each supported QDoc request and its QAD Enterprise Applications calling program is associated with an XML schema file. See Chapter 24, “QDoc Specifications and Standards,” on page 275 for details.

The XML schema for a calling program such as Sales Order Maintenance (`sosomt.p`) contains all possible data entry fields in the calling program. It also includes the information required to start data iterations, which are repetitive entry sequences such as sales order lines. For information on schema exceptions, see “QDoc Exceptions” on page 362.

For example, the schema document for `sosomt.p` in QAD EE and QAD SE is named `maintainSalesOrder-2008_2.xsd`. and contains calling procedure as well as detailed field information.

Each QDoc response also has an XML schema file. It is named similarly to the request QDoc schema, but includes `response` in the name. For `sosomt.p` in QAD EE and QAD SE, the response schema file is named `maintainSalesOrderResponse-2008_2.xsd`. This file contains procedure information as well as a list of primary key fields for each iteration. You can edit the schema as required to filter out or add QAD Enterprise Applications field values that will be included in the response document.

For additional information on configuring response document contents, see “QDoc Response Data” on page 154.

QDoc XML Schemas (1.0)

Important Schemas using the 1.0 specification are supported for backward compatibility only. Where possible, you should use the 1.1 schemas, since these are WSI-compliant.

In most cases, two XML schema files are associated with each supported QDoc request and its QAD Enterprise Applications calling program when using the 1.0 specification.

The XML schemas for Sales Order Maintenance (`sosomt.p`) contain all possible data entry fields in the calling program and information required for data iterations. For example, the schema documents for sosomt.p in eB2 are:

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

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

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

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

The first file is the base schema file, and contains calling procedure information. The second file, the type file, contains a list of primary key fields for each iteration. Editing the type file enables you to filter out or add QAD Enterprise Applications field values that will be included in the response document.

QDoc Events Documents

The QDoc—if using the user interface API—also needs information on how to navigate in an QAD Enterprise Applications screen. By default, the QDoc sends data and moves to the next field. However, it needs to know, given the field location, when to accept an iteration—as for multiple sales order lines within a single sales order—and when to allow a deletion. These screen actions are called events and are stored in a version-specific directory for each calling program in an events document named by default after the calling program. The default events document for `sosomt.p` is `sosomt-ERP3_2.xml`. The base events documents are also available on the QAD Web site. For information on event exceptions and errors, see “Event Exceptions” on page 353.

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

Custom QDocs

You can create QDoc XML schema and events documents for programs you have created or customized using a tool called QGen that ships with QXI. QGen creates schema files for request and response documents, as well as an events file. See Chapter 18, “QGen,” on page 215.

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

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

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

Transforming Non-Standard XML Documents

Typically, the QXI Queue Manager expects all incoming documents to be in proprietary QDoc format. The Queue Manager can add a valid SOAP envelope if one is missing. It can also accept non-standard XML documents and transform them into QDocs with a valid SOAP envelope using the QXI transformation engine. See “QXI Transformation Engine” on page 166 for details.

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

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

WSDL Document for Web Service Clients

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

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

You can generate WSDL files from the QXI Receiver configuration page (for details see “Inbound Receivers” on page 180). For every WSDL file it will include request and response schema files. So before using the WSDL file, make sure the WSDL file—and the request and response schemas files—are in the same directory.

QXtend Code Page Support

QXI supports the use of multiple code pages across the products linked by QXtend. The products involved are Progress, Java, QXI, and QAD Enterprise Applications. Each has an entry in the `dtencode.dat` file shipped with QXI. This file is located in *`QXtendInstallDir`*.

Supported QXtend Integrations

QXI is either required or available for certain QAD products. These include QAD Production Scheduler and QAD Mobile Field Service.

For each of these products, use *Installation Guide: QAD QXtend* to install QXI, then refer to the appropriate section in the guides for these products to complete the specific configuration requirements for QXI.

Chapter 14

Configuring and Using QXtend Inbound

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

Introduction

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

Use the Configuration Manager to manage your QDoc schemas, to create and modify your QAD Enterprise Applications receivers, and to allocate QDoc schemas to the receivers. Use the Connection Pool Manager to create, view, and control the available telnet and appserver connections to your QAD Enterprise Applications sessions.

An additional interface that is not a required component of QXI is the Queue Manager. This tool provides a QDoc queuing service, creating request and response directories for each external application that is sending QDoc requests. The request QDocs are picked up from a specified request directory; the response QDocs are returned by the Queue Manager to a specified response directory where the external application picks them up. See Chapter 15, “QXtend Inbound Queue Manager,” on page 163.

QDoc Response Data

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

For additional information on specific exceptions included in response QDocs, see Chapter 28.

Default Warnings and Errors

Additional warnings and errors are:

- Notifications from pre- and postprocessing programs
- Occurrences of unused fields in QAD Enterprise Applications
- Occurrences of unused iterations in QAD Enterprise Applications
- Occurrences of overlength field data in QAD Enterprise Applications

Pre- and Postprocessing Messages

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

For details on how to make the correct calls to add these messages to the response, see “Warnings and Errors” on page 236.

Messages for Unused Fields

In some cases, fields may exist in a QDoc that are never used during the processing of that QDoc. For example, if a field node exists in the QDoc that is not part of the schema or is misspelled, it is not matched with a field in QAD Enterprise Applications and remains unused during the

processing of that QDoc. This information is added to the response QDoc as a warning regardless of the suppressResponseDetail setting. For details see “Include QAD Enterprise Applications Field Values” on page 156). The response message is:

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

Messages for Unused Iterations

In some cases, entire iterations in a QDoc are never used during the processing of that QDoc. For example, a control program disables the display of a frame but an iteration with fields for that frame is included in the QDoc. Since that frame is never visited during the processing of the QDoc, a warning displays in the response QDoc regardless of the `suppressResponseDetail` setting. For details see “Include QAD Enterprise Applications Field Values” on page 156).

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

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

Controlling Messages for Unused Fields and Iterations

You can control the display of messages arising from unused fields and iterations in response QDocs by using the `excludeUnusedWarnings` node. By default, the display of these messages is suppressed.

To include messages for unused fields and iterations in the response documents, edit the `qxtendconfig.xml` file located in *`TOMCAT_HOME`*`/webapps/<`*`QXI webapp`*`>/WEB-INF/conf`. Locate and set the following node to false; for example:

```
boolean excludeUnusedWarnings = “false”
```

Messages for Field Length

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

```
Data submitted to QAD Enterprise Applications was too long for the QAD
Enterprise Applications field size.
```

Include Java Trace Information

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

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

To include the Java trace information, change this to:

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

Include QAD Enterprise Applications Field Values

By default, the QDoc response document contains the primary key values from each iteration (for example, sales order number and sales order line item values). If using the 1.0 syntax specification, you can edit the response type schema to eliminate or add field values you want to see. For the 1.1 syntax specification, edit the response schema.

However, by default, no values are returned in the response document. To have values returned, edit the specific QDoc according to the syntax specification in use.

In the 1.0 syntax specification, locate the setting `suppressResponseDetail = “true”` in the body of the document and change it to `suppressResponseDetail = “false”.`

In the 1.1 syntax specification, locate the `suppressResponseDetail` node in the `ReferenceParameters` section of the SOAP header:

```
<suppressResponseDetail xmlns="urn:schemas-qad-com:xml-
services:common">true</suppressResponseDetail>
```

and change the value of the node to false:

```
<suppressResponseDetail xmlns="urn:schemas-qad-com:xml-
services:common">false</suppressResponseDetail>
```

For the 1.0 syntax version, to control which field values are returned, create a copy of the response type schema file (`salesOrderResponseType-eB2_1.xsd` for `sosomt.p`) for the specific response document and edit it. (For the 1.1 syntax version, create a copy of the response schema.) Comment out the fields you do not want to see in the response document, and add in any fields you want to add, then save it as a custom schema file. You then assign the response schema file to the master schema lists where it can be assigned to a specific receiver. See “Adding a Schema to the Master Lists” on page 189 for steps.

Note To add fields not already referenced in the response schema, you can copy them from either the request type schema (for the 1.0 syntax) or from the request schema (for the 1.1 syntax).

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

1. Open *`TOMCAT_HOME`*`/webapps/<`*`QXI webapp`*`>/WEB-INF/conf/qxtendconfig.xml.`
2. Locate the `messageServletURL` attribute within the `<general-config>` node.
3. Enter the correct host and port values to communicate with the QXI server.

   ```
   <messageServletURL label=“Message Servlet URL” value=“http:
   //<host>:<port>/<QXI webapp>/MessageReceiverServlet”/>
   ```

4. Save and close the file.

Performance Considerations

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

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

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

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

Starting QXtend Manager

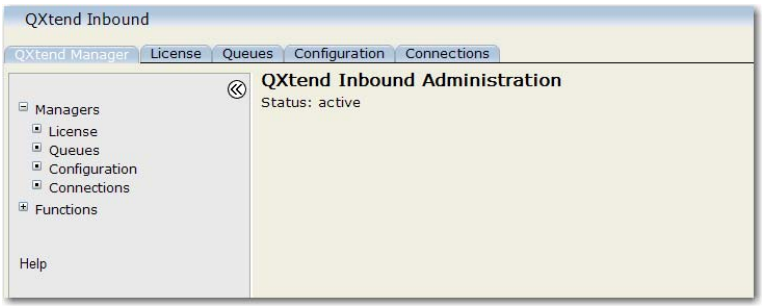
Prior to starting the QXtend Manager, start Tomcat and your target QAD Enterprise Applications sessions. Then open a browser and enter the following URL:

`http://<hostname>:<tomcat_port>/<QXI webapp>/`

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

The QXtend Manager opens.

Fig. 14.1
QXtend Manager



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

The QXtend Manager functions are:

Restart Server. This shuts down QXI and restarts it, using the Tomcat restart link.

Install Log. This displays `qdocInstall.log`. The log is updated each time the QXI Web application is started, stopped, or reloaded. The log is stored in:

TOMCAT_HOME`/webapps/<`*QXI webapp*`>/WEB-INF/logs`

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

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

Resume. This restores QXI services following administrative changes.

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/<`*QXI webapp*`>/WEB-INF/conf`

All log files are created and stored by default in:

TOMCAT_HOME`/webapps/<`*QXI webapp*`>/WEB-INF/logs`

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

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

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

Log Report Levels

Reporting levels for the QXI logs are:

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

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

The logs created in QXI are:

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

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

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

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

qdocInfo.log

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

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

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

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

These files are rolled over daily.

queue.log

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

connectionPools.log

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

qdocInstall.log

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

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

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

qxtendserver.log

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

transformationEngine.log

This log file contains information about key steps in the transformation, namely creating managers. It contains error information when a problem occurs. When set to debug it shows detailed transformation steps. For details, see “QXI Transformation Engine” on page 166.

transformationengineRequests.log

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

transformationengineRequests.log

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

transformationEngine.debug

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

Testing QXI Processes

QXI provides a tool set for testing QXI processes.

Process Request

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

TOMCAT_HOME`/webapps/<`*QXI webapp*`>/WEB-INF/receivers/<`*receiverName*`>/requests`

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

Note The Process Request option supports QDocs in both the 1.1 and 1.0 syntax specifications. A one-time configuration during install determines which QDoc standard to use when displaying the QDoc. For details see *Installation Guide: QAD QXtend*.

Create Empty QDoc

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

Note For UI API schemas this option drops the parent key from the child iteration. The created QDoc contains only those fields that need updating on the screen.

Verify QDoc Supported

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

Verify Receiver

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

UI Adapter Connection Test

This option lets you test the UI adapter connection by entering a receiver. This avoids having to create QDocs that contain dummy data.

Note The UI Adapter Connection Test function can only display a summary of a QDoc response for QDocs that use the 1.1 syntax specification.

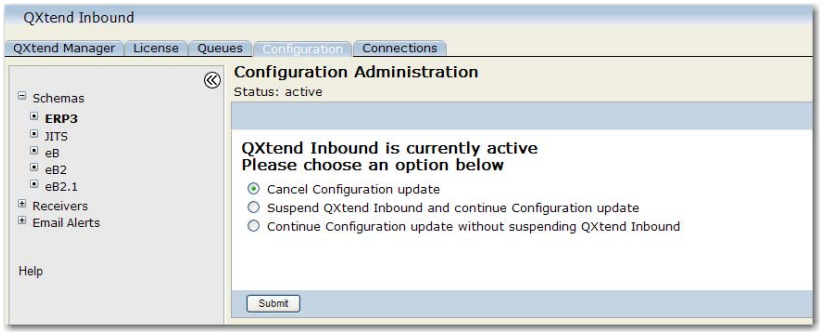
In the Receiver field, enter the name of the QAD Enterprise Applications session you want to connect to—QAD SE, for example. If you employ a user ID and password to connect to the target QAD Enterprise Applications instance, you must complete the fields in the Requestor Details section; otherwise, this information is optional. A message displays stating whether or not the QDoc was successful.

Suspending and Resuming Processing

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

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

Fig. 14.2
Suspension Options Prior to a Schema Update



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

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

You can also choose Suspend from the QXI Administration page.

Resume QXtend

To resume processing, choose Resume from the QXI Administration page.

Chapter 15

QXtend Inbound Queue Manager

The following material provides an overview of using the QXtend Inbound (QXI) Queue Manager.

Introduction

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

The Queue Manager is accessed through the QXtend Manager interface, but is not a required or default component of QXI. QXI accepts API requests directly as HTTP post requests in SOAP-compliant XML format, or can accept the documents from a queuing application such as IBM MQSeries, or from the QXtend Queue Manager. You can also use QAD Q/LinQ for a more robust queuing and administrative application.

See Chapter 21, “QXtend Inbound with QAD Q/LinQ,” on page 245.

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

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

Note The Queue Manager supports both the QDoc 1.1 and 1.0 syntax specifications, and the SOAP 1.1 standard.

Non-standard XML requests can also be received by queues set up to allow them. These are XML documents added to the queue that do not have a `.req` extension. In this case, the document is passed to the transformation engine. The engine applies a custom XSLT style sheet based on the extension of the file and creates a standard QDoc request. It places this request in a standard queue directory where it is processed normally. For details, see “QXtend Inbound Queue Manager” on page 163.

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

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

ok. The QDoc processed correctly.

.wrn. The QDoc processed but encountered warnings.

.err. The QDoc request failed.

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

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

Multi-Threaded and Single-Threaded Queues

Multi-threading QDoc queues allows a higher throughput and better performance. However, in multi-threaded queues where QDocs have a sequential dependency, it is possible for a dependent QDoc to start processing before the initial QDoc has finished. Therefore, queues are defined with thread sizes to allow you to create multi-threaded queues for QDocs without dependencies; single-threaded queues for QDocs with dependencies. See “Thread Settings” on page 176.

Initializing the Queue Manager

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

1. Open `environmentmanager.xml` in:

   TOMCAT_HOME`/webapps/<`*QXI webapp*`>/WEB-INF/conf`

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

   For information on queue errors, see “Queue Exceptions” on page 357.

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

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

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

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

5. Save the file.

6. Restart QXI in order for the changes to take effect.

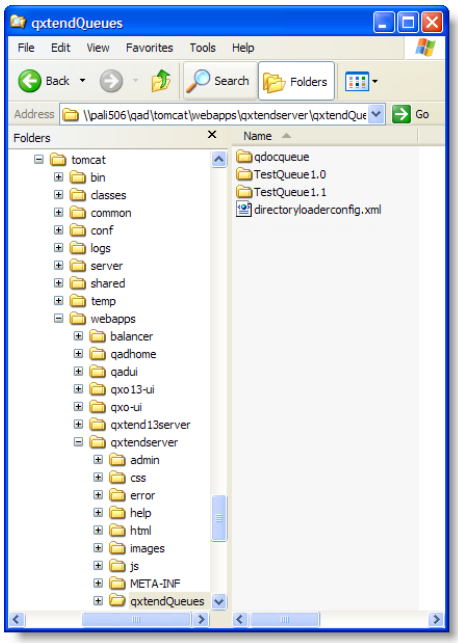
Important It may be necessary to stop and restart Tomcat before the Queue Manager displays.

Queue Manager Directory Structure

The default Queue Manager directory structure is located under:

TOMCAT_HOME`/webapps/<`*QXI webapp*`>/qxtendQueues`

Fig. 15.1
Queue Manager Sample Directory Structure



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

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

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

Queue Manager Logs

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

`TOMCAT_HOME``/webapps/<`*`QXI webapp`*`>/WEB-INF/logs`

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

The error level is controlled by modifying the level value for the logger name directoryloader in `qxtendlogging.xml`. See “Log Report Levels” on page 158 for information on reporting levels.

QXI Transformation Engine

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

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

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

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

Create an XSLT Mapping Specification

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

TOMCAT_HOME`/webapps/<`*QXI webapp*`>/WEB-INF/mappingSpecs`

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

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

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

Creating a Request Parser

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

To create a new request parser, follow these steps:

1. Create the following directory:

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

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

2. Edit the existing request parser program to put in your own parsing code:

   `WEB-INF/RequestParser.java`

3. Compile your `RequestParser.java` program into `RequestParser.class`.

4. Copy `RequestParser.class` into the directory created in step 1.

5. Modify it for your proprietary file format.

6 To configure the new request parser, add an entry to `requestparsermanager.xml` as shown in “requestparsermanager.xml Changes” on page 168.

7 Add the file extension mapping to the document standard used by the new request parser in `directoryloaderconfig.xml` as shown in “directoryloaderconfig.xml Changes” on page 168.

Modifying QXtend Configuration Files

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

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

directoryloaderconfig.xml Changes

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

```
<directoryloaderConfig>

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

...

<directoryloaderConfig>
```

requestparsermanager.xml Changes

Add the document standard to `requestparsermanager.xml`:

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

transformationmanager.xml Changes

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

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

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

```
        </contentTypeMaps>

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

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

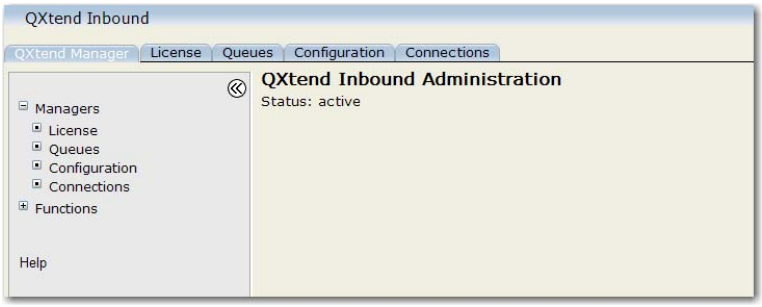
Starting the Queue Manager

Access the Queue Manager through the QXtend Manager at:

`http://<hostname>:<tomcat_port>/<QXI webapp>/`

This assumes Tomcat and the target QAD Enterprise Applications instances are running.

Fig. 15.2
Queue Manager Main Menu



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

Using the Queue Manager

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

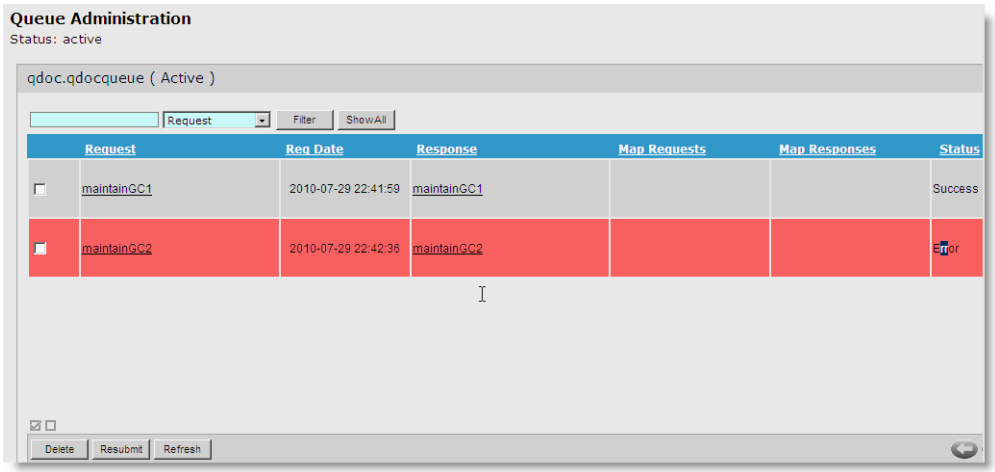
View Individual Queues

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

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

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

Fig. 15.3
Set Number of Documents in Queue to Display



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

Sorting

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

Filtering

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

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

Choose Show All to display all queues again.

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

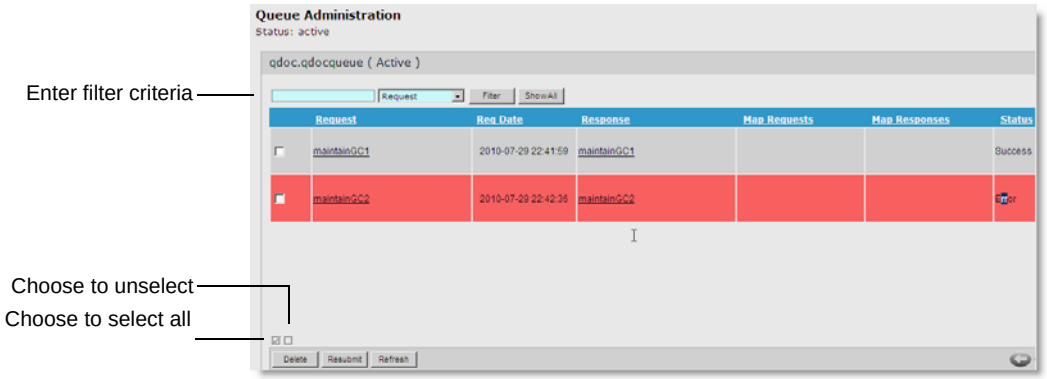
Deleting

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

Resubmitting

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. See “Edit Failed Submissions” on page 172.

Fig. 15.4
Error and Success Documents in Queue

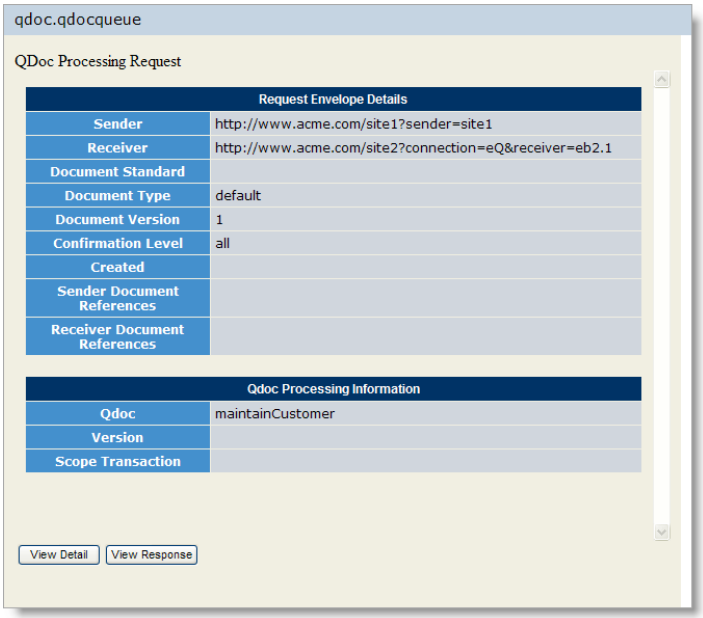


In the lower left-hand corner above the Delete button are select-all and select-none symbols. Choose the select-all symbol—the box with a checkmark inside—to select all documents in the queue; choose the empty box to unselect all documents.

Viewing Documents

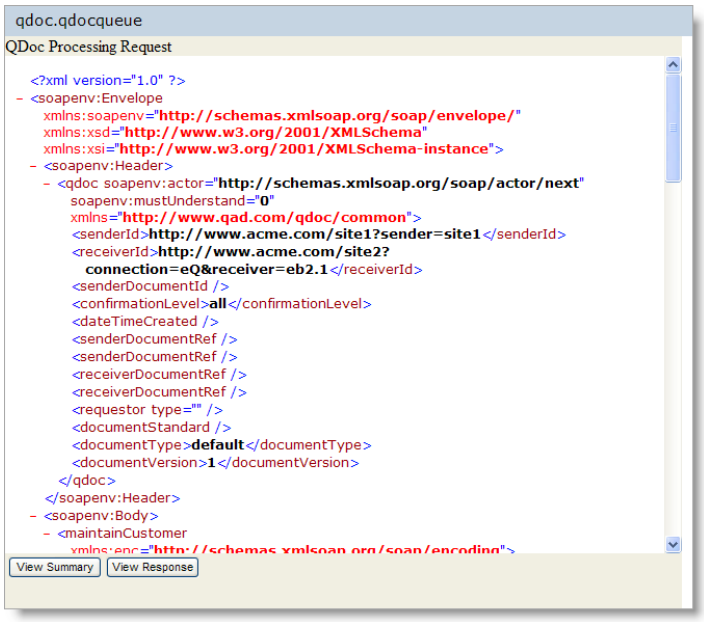
For any request or response document, choose the document name in the queue display to view the document. A summary screen displays.

Fig. 15.5
Summary View of a Request Document



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

Fig. 15.6
Detail View (XML) of Response Document

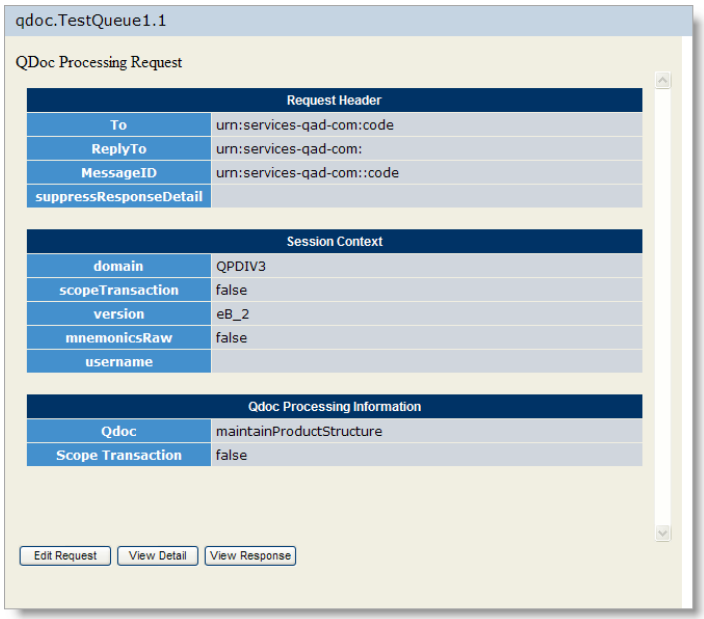


Edit Failed Submissions

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

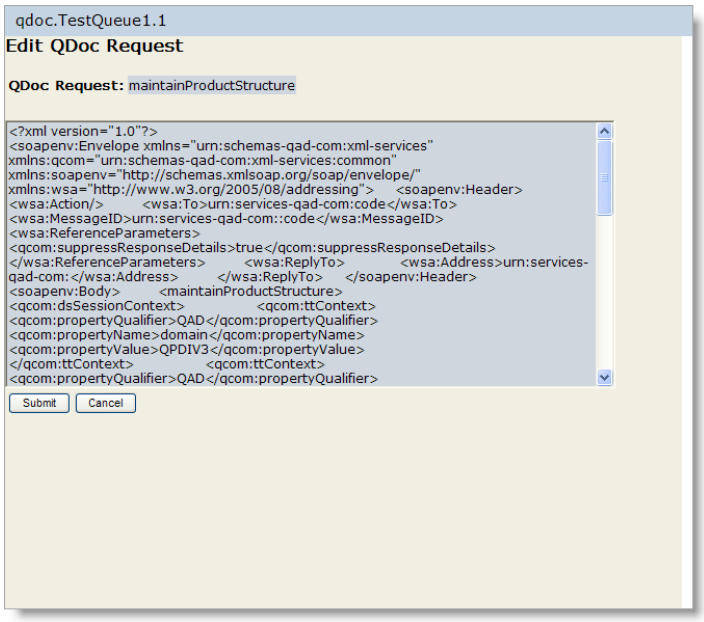
1 Select either the request or response document name in the queue. The summary view opens.

Fig. 15.7
Summary View of an Error Status QDoc



2 Choose Edit Request to open the XML editor.

Fig. 15.8
XML Editor for Error Status QDocs



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

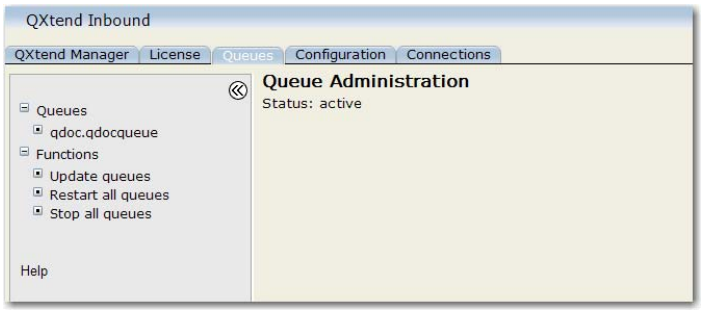
4 Choose Submit. The message “Request re-submitted successfully” displays. The document is immediately updated in the queue. Review the related response document for further error checking.

Queue Manager Functions

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

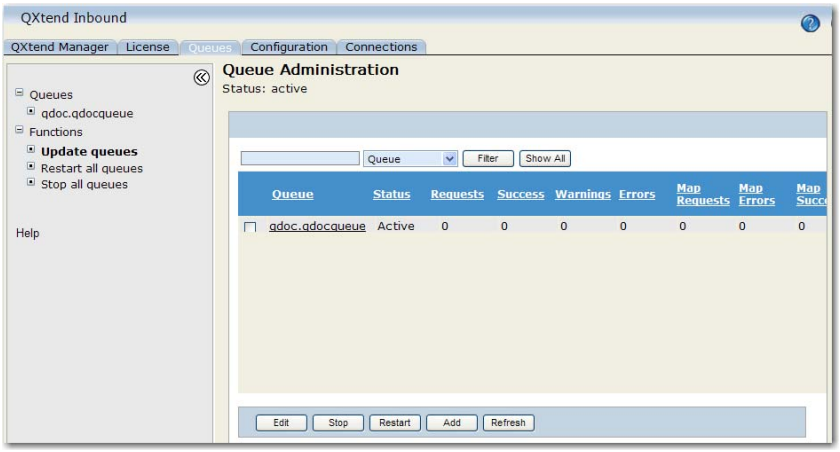
- Update queues
- Restart all queues
- Stop all queues
- Add a queue
- Modify a queue

Fig. 15.9
Queue Manager Functions Menu



When you select Update Queues under the Functions menu, a list of existing queues displays. You can filter and sort this list as described under “Sorting” and “Filtering” on page 170.

Fig. 15.10
Update Queue



Stop or Restart Queues

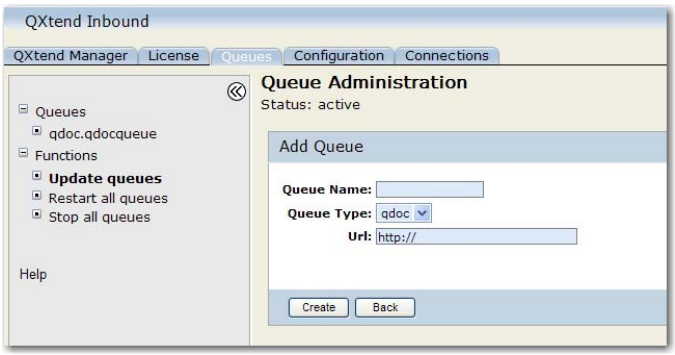
1 To stop or restart one or more queues, choose the checkbox next to the queue names you want to stop or restart.

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

Add a Queue

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

Fig. 15.11
Add a Queue



2 Enter the following:

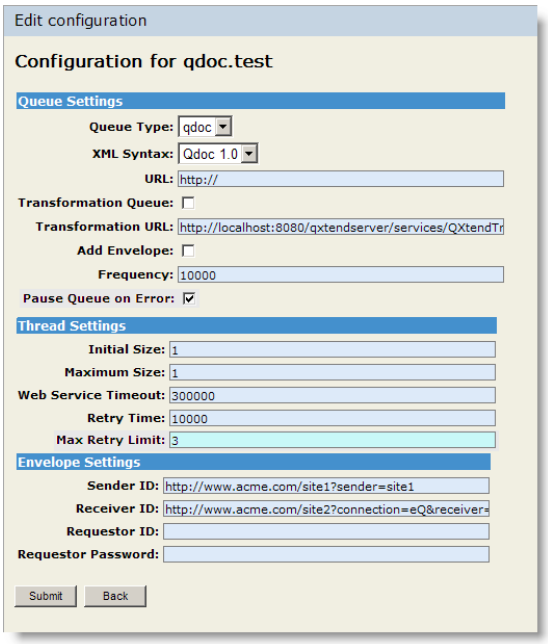
Queue Name. You can enter any name here. The final queue name will be <*queue type*>.<*your queue name*>. So if you enter `testQ` here, the queue name becomes `qdoc.testQ`.

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

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

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

Fig. 15.12
Edit Queue



Queue Settings

Update the fields using the details that follow:

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

XML Syntax. Select the XML syntax specification to use for the queue: Qdoc 1.0 or Qdoc 1.1. Only required if the Add Envelope check box is selected.

URL. URL is the location of the QXI Web Services component.

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

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

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

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.

Yes: Queue Manager stops on any error. When restarted, Queue Manager resumes processing the QDoc that paused the queue.

No: When error occurs, Queue Manager proceeds to the next QDoc without pausing the queue.

Thread Settings

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

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

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

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. These options change depending on the XML syntax version that is selected. Depending on the settings, the correct SOAP envelope is created.

Use the following setting if the XML syntax is 1.1.

Receiver. This identifies the QDoc recipient, which is by definition an QAD Enterprise Applications instance. Enter the name of a receiver defined in the QXI system.

Use the following settings if the XML syntax is 1.0.

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

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

```
sender=site1
```

Receiver ID. The receiver is defined as a URL. At a minimum the Receiver ID must include the receiver parameter. To uniquely identify a receiver, add an QAD Enterprise Applications instance designation:

```
receiver=eB2_Prod
```

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

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

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

Modify a Queue

1. To update a queue configuration, select the checkbox next to the queue you want to edit.
2. Choose Edit. The Edit Configuration screen displays. Make changes as described in step 2 under “Add a Queue” on page 174.
3. When the queue setup is complete, choose Submit. The message “Queue Config saved successfully” displays.

Queue Manager Scripts

This script makes use of GNU sed and GNU wget so that it works on most flavors of UNIX, including HP-UX and AIX. So before running this script, please install the GNU sed and GNU wget, which are freeware downloads available from http://ftp.gnu.org/gnu/. Once both sed are gnu are installed, copy the executables (sed and wget) from the directory you installed (default `/usr/local/bin`) to this scripts directory to make sure the GNU versions of the programs are running and not the native OS ones.

In addition to managing queues through Queue Manager in the administrative UI, you can also use the queue manager script to start, stop, restart, and shut down queues, as well as query the queue status:

```
queue-control.sh -Operation [Queue]
```

Where *`Queue`* is the connection queue name.

Operation	Description
h	Print the script help
q	Query the queue status
r	Restart the queue
s	Start the queue
x	Stop the queue
t	Shut down the queue

Example Use the following command to stop the queue named qdoc.MyQueue:

```
./queue-control.sh -r qdoc.MyQueue
```

Chapter 16

QXtend Inbound Configuration Manager

The QXtend Inbound (QXI) Configuration Manager provides a toolset to manage standard QAD and custom schemas and the QAD Enterprise Applications receivers.

Introduction

The Configuration Manager provides an interface to manage QDoc schemas, create and modify QAD Enterprise Applications receivers, and allocate QDoc schemas to the receivers.

Access the Configuration Manager through the QXtend Manager at:

`http://<hostname>:<tomcat_port>/<QXI webapp>/`

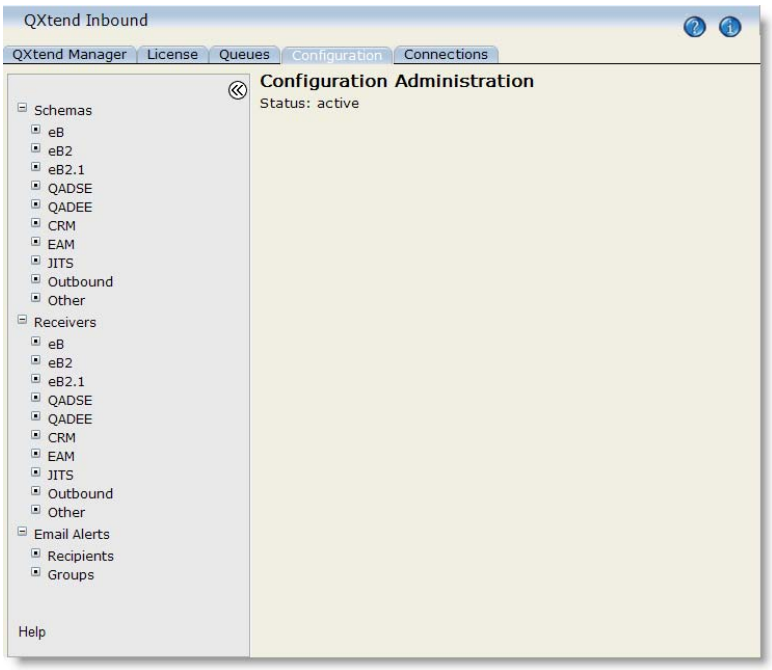
For details about schemas see “QDoc Schemas” on page 267.

This assumes Tomcat and the target QAD Enterprise Applications instances are running.

The Configuration Manager is launched from the QXI interface and lets you manage your master list of available QDoc schemas and set up and configure your QAD Enterprise Applications receivers.

For information on configuration errors, see “Configuration Exceptions” on page 368.

Fig. 16.1
Configuration Manager Interface



Inbound Receivers

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

In the Configuration Manager, you can:

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

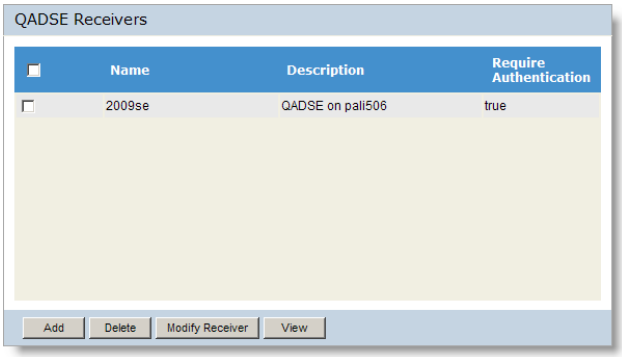
- Remove standard or custom schemas from receivers.
- Create WSDLs for a receiver/API combination.
- Delete receivers.

Adding Inbound Receivers

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.

2 The currently available receivers display for that version.

Fig. 16.2
Receiver View in Configuration Manager



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

4 Enter a name for the receiver.

Fig. 16.3
Add Receiver

5 Select the Require Authentication check box to indicate that this receiver requires all QDocs to contain authentication information.

In previous versions QXI used the system-wide `UseQDocRequestor` setting. For each receiver in the system, you can specify that every QDoc that is for a receiver must contain authentication credentials consisting of a username and password, or a session ID.

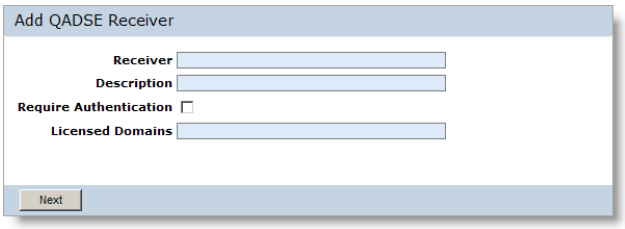
For details about `UseQDocRequestor` see “Requestor Element” on page 302.

6 Select the Licensed check box to indicate whether the receiver will be processing requests that require a licensed agent.

7 If the schema is for eB2.1 or later, you can define a list of domain codes for the new receiver. The License Manager will reject any non-free or charged message/QDoc for an undefined domain.

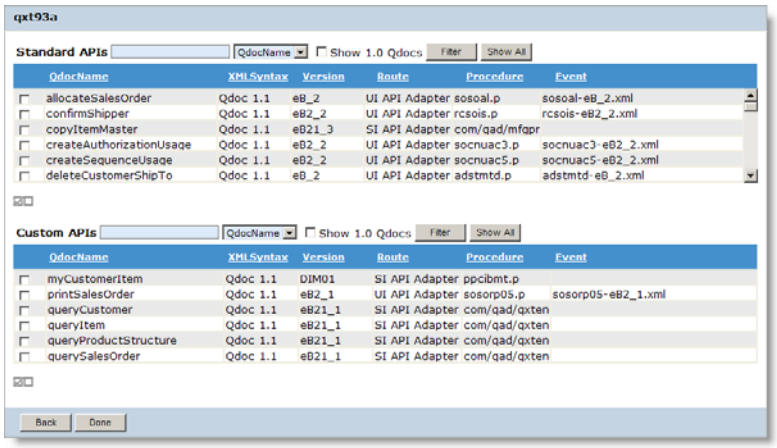
See “QAD QXtend Licensing” on page 253 for details on licensing.

Fig. 16.4
Naming a Receiver



8 Choose Next. The list of available QDoc APIs displays.

Fig. 16.5
Adding QDoc APIs to a New Receiver



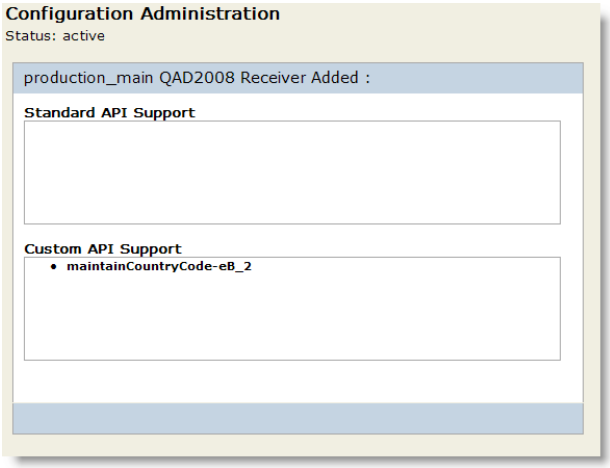
9 Select from the available lists of standard and custom QDocs by clicking the QDocs you want.

10 Choose Done to make the assignment. A confirmation screen displays.

Note A receiver must be associated with an API that uses the SI API adapter route in order to receive business objects that employ DDP. The service interface adapter API is used by the DDP API to input data into the target application.

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

Fig. 16.6
QDocs Added to a New Receiver



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

Adding a Schema to an Existing Receiver

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. See “QDoc Schemas” on page 187.

2 Select the receiver to which you want to add a schema by clicking its box and then click Modify.

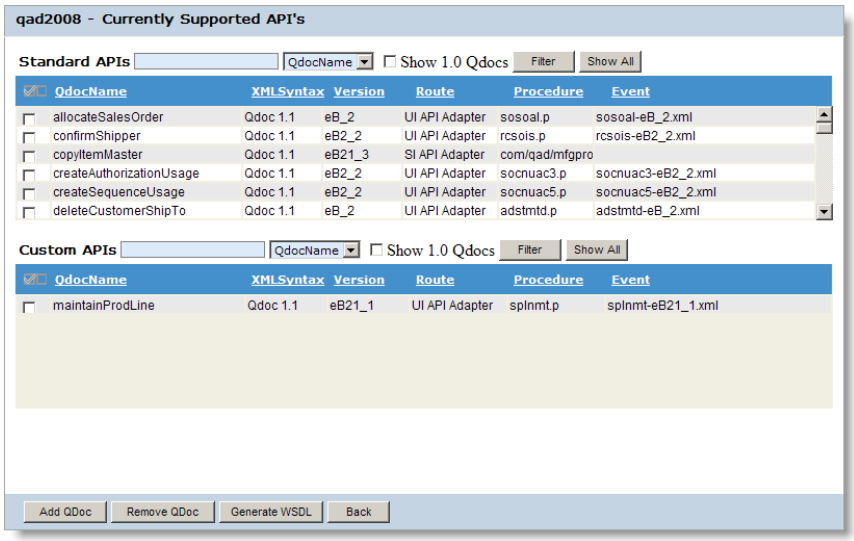
3 Select the appropriate Suspend option if QXI is active and then click Submit.

4 The Modify <*application*> Receiver dialog displays, which allows you to change description, authentication, and licensed domains attributes.

5 Make any required changes and then click Next. A list displays of the standard and custom APIs that are currently supported by the receiver. Click Next.

Note To view schemas that use the 1.0 specification, select the Show 1.0 QDocs check box. By default, this screen only displays schemas that use the 1.1 specification.

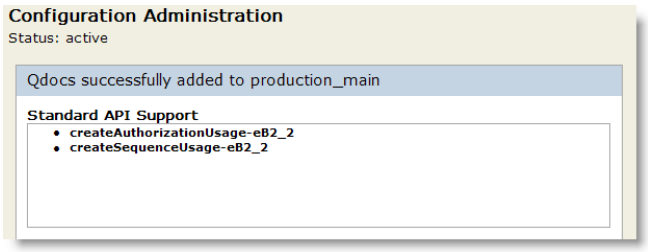
Fig. 16.7
Adding a QDoc to an Existing Receiver



6 Select the schema or schemas to add to the receiver.

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

Fig. 16.8
QDocs Added to an Existing Receiver



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

Removing a Schema from an Existing Receiver

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 display.

2 Select the receiver from which you want to remove a schema by clicking its box and then click Modify.

3 Select the appropriate Suspend option if QXI is active and then click Submit.

4 The Modify <*application*> Receiver dialog displays, which allows you to change description, authentication, and licensed domains attributes.

5 Make any required changes and then click Next. A list displays of the standard and custom APIs that are currently supported by the receiver. Click Next.

6 Select the QDocs to remove from the receiver by selecting the associated check boxes. Click Remove QDoc.

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

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

Generating WSDLs for a Receiver/API

You can generate a WSDL for an API—or set of APIs—for a given receiver/API combination for APIs that use the 1.1 syntax specification. For each API selected, one WSDL is generated along with the associated request schema and response schema.

When you generate WSDL for a set of APIs, the WSDLs are returned as a compressed archive in a `QdocWebService.zip` file bundled with the associated `.xsd` files. The `.zip` file can be saved to a user-defined location on the client. When you generate a WSDL for a single API, the resultant compressed `.zip` file uses the same name as the API name.

A sample empty Qdoc is available for each API in WSDL generation.

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. See “QDoc Schemas” on page 187.

2 Select the receiver to generate a WSDL for by selecting its check box and then click Modify.

3 Choose the appropriate Suspend option if QXI is active. Click Submit.

4 The Modify <*application*> Receiver dialog displays, which allows you to change description, authentication, and licensed domains attributes. Make any required changes and then click Next.

5 A list displays of the standard and custom APIs that are currently supported by the receiver.

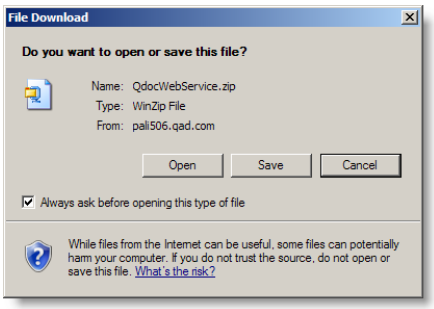
6 Select the APIs for which you want to generate a WSDL. You can select any combination of standard APIs, custom APIs, or both.

7 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.

8 The WSDL is generated and a File Download box displays.

Fig. 16.9
File Download



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

The `.zip` file contains the following files:

- `QdocWebService.wsdl` - If combining multiple WSDLs, this file contains all selected API operations.
- `[API Name]-[API version].wsdl` - If not combining all WSDLs or if just selecting one API, the file contains a WSDL file for every API.
- `[API Name]-[API version].xsd` - This file contains as many request schema files as there were API operations that were not excluded. These are refactored schemas, not those stored on the QXI server, so that they all work when imported into the same XML namespace.
- `[API Name]Response-[API Version].xsd` - As many response schema files as there are request schemas with a corresponding response schema. The WSDL will specify that the response schema is the same as the request schema, if no response schema can be found.
- `exclusions.txt` - A text file containing a list of all the selected APIs that were not included in the WSDL and the reason they were not included. Reasons for exclusion may include the following:
  - The API is not for QDoc version 1.1.
  - A request schema file could not be found.
  - A different version of an API of the same name has already been added (WSDLs can only have one operation with a given name). The first operation that was added will be included.
  - An error occurred when trying to add an operation for the API to the WSDL.
  - A schema did not validate for the reason given.
- `[API Name]-[API version]-Sample.xml` - This is a sample QDoc file.

Deleting a Receiver

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 display.

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

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

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

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

`TOMCAT_HOME/webapps/<QXI webapp>/WEB-INF/deletedReceivers`

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

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

QDoc Schemas

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

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

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

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

Multiple Events Files

You can also maintain multiple events files for a given schema. By default, events files are named like the QAD Enterprise Applications program name (omitting the `.p` extension) appended with the QDoc version and an `.xml` extension. For example, the menu-level program `sosomt.p` would have a default events file of `sosomt-ERP3_2.xml`.

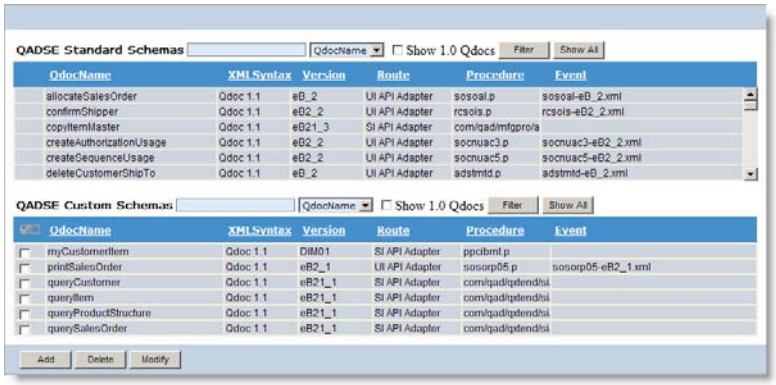
You can create multiple events files to support different paths through a program, as for example, when a site is using advanced pricing for some orders and not for others.

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

View QDocs by QAD Enterprise Applications Version

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

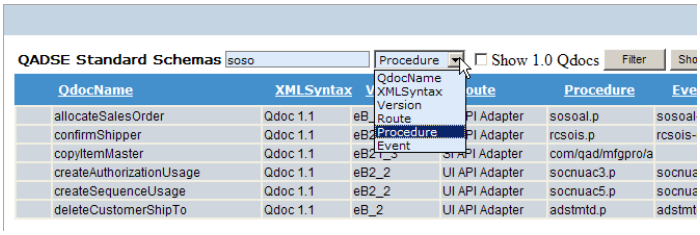
Fig. 16.10
Schema View in Configuration Manager



Note To view schemas that use the 1.0 specification, select the Show 1.0 QDocs check box. By default, this screen only displays schemas that use the 1.1 specification.

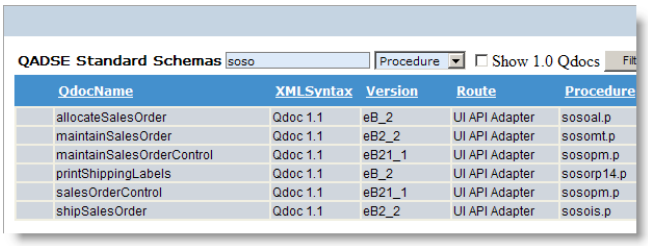
2 To limit the schema list by a text string criteria—such as all QDocs that contain “soso” in the procedure name—enter the text string in the field above the list and select the column to filter from the drop-down list.

Fig. 16.11
Entering Filter Criteria for Schemas



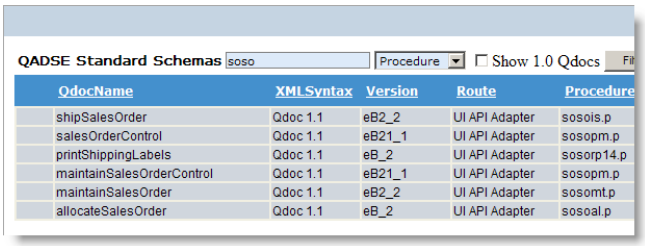
3 Choose Filter to see the results. Choose Show All to clear the filter and view all documents.

Fig. 16.12
Filtered Schema Results



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

Fig. 16.13
Sorting Schemas



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

Adding a Schema to the Master Lists

Schemas are the QDoc schemas and events files generated using QGen.

For information on QGen, see Chapter 18, “QGen,” on page 215.

QAD ships standard QDoc APIs. Additional standard QDocs may be made available on the QAD Web site at:

http://support.qad.com/

You can add schemas using either the 1.1 or 1.0 QDoc syntax. You specify a request type or response type for the schema only if using the 1.0 syntax specification.

You also can generate schemas and events files for customized or custom Progress programs.

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

```
TOMCAT_HOME/webapps/<QXI webapp>/
WEB-INF/schemas/<mfgpro-version>/
```

Custom schemas are added to:

```
TOMCAT_HOME/webapps/<QXI webapp>/
WEB-INF/schemas/<mfgpro-version>/custom
```

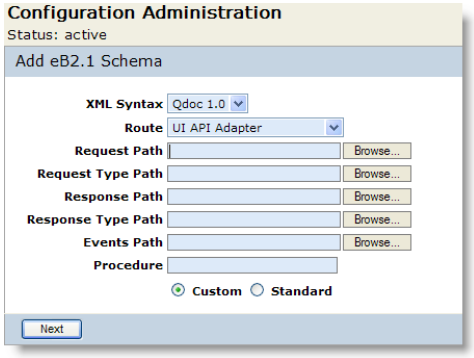
When adding new QDocs, you can select which receivers to add them to, and the new QDocs are available to be added to any new receiver you create in the future.

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

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

2 Click Add and then select the appropriate Suspend option if QXI is active. The Add Schema screen displays.

Fig. 16.14
Adding a Schema



3 Enter the following information:

XML Syntax. Specify the XML syntax version to use for the schema.

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

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

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

Request Type Path. (QDoc 1.0 syntax only). Specify the full path to the request type schema, typically the same path as for the base schema; for example:

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

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

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

Response Type Path. (QDoc 1.0 syntax only). Optional. This path and file name are required only when you want to specify a response type schema. Enter the path to the base response type schema that is to be uploaded to the standard or custom schema directory in Tomcat; for example:

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

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

```
TOMCAT_HOME/webapps/<QXI webapp>/
WEB-INF/events/mfgpro_version
```

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

```
TOMCAT_HOME/webapps/<QXI webapp>/
WEB-INF/events/eb2/downloads/sosois-eB2_1.xml
```

Procedure. Specify the target procedure name in QAD Enterprise Applications, such as `customProgram.p` or `sosois.p`.

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

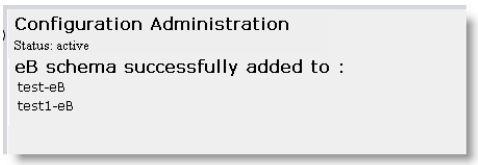
4 Choose Next. If the QDoc already exists, you are given the option to overwrite the existing QDoc. If you overwrite, you continue and the Add Receiver Support page displays. If you do not overwrite, you are returned to step 2 and must re-enter the information.

Fig. 16.15
Adding a Schema to Existing Receivers

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

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

Fig. 16.16
Confirming a Schema Addition



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

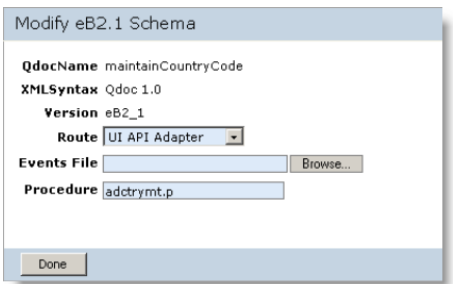
Modifying a Schema Configuration

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

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

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

Fig. 16.17
Modify a Schema Configuration



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

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

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

Delete a Custom Schema Configuration

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

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

2 Select the custom schema you want to delete and choose Delete and then choose the appropriate Suspend option if QXI is active.

3 You are asked to confirm. Choose OK. A confirmation displays.

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

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

QXI E-mail Alerts

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, which are triggered by QXI at a stage in processing that does not yet involve a specific receiver, domain, or API. Default alerts may arise due to situations such as unused fields in QAD Enterprise Applications, or if a QDoc request is not well-formed—for example, there may be a missing requestor node.

  Note For details on default warnings and errors, see the section “Default Warnings and Errors” on page 154.

- Processing alerts, which are triggered by processes involved in handling QDocs. For example, processing alerts may arise due to the faulty configuration of a receiver.

System alerts and default alerts are typically used by system administrators to facilitate troubleshooting.

Using the e-mail alert feature in QXI you can register recipients and receivers to an alert group, domains to a group/receiver, and APIs to a group/receiver/domain. This flexibility allows e-mail alerts to be distributed to recipients to fulfill various notification purposes:

- Receiver registration provides the ability to receive alerts for specific QAD Enterprise Applications instances. For example, alerts can be distributed for events that only occur within QAD SE.

- Domain registration provides the ability to receive alerts for specific sites or geographical locations. Alerts can be distributed to personnel with responsibility for different areas or regions of an organization.
- API registration provides the ability to receive alerts for a specific business area—certain APIs might be associated with sales orders, for example. Alerts can be distributed to personnel within a sales order administration group.

Similar e-mail alert functionality—tailored to event and QDoc processing—is available in QXO.

In the QXI Configuration Manager you can:

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

Configuring E-mail Settings

Use the Email Configuration Parameters screen to configure e-mail settings. You can validate your e-mail setup by clicking the Verify button. The process for configuring e-mail settings in QXI is the same as that in QXO.

Note Before configuring e-mail settings, ensure that access is available to the SMTP server.

1 Click the Configuration tab.

2 Click the Email Alerts node in the left-hand tree view. If it was not already displayed, the Email Configuration Parameters screen displays.

Fig. 16.18
Email Configuration Parameters

3 Enter the following values as required.

SMTP Server. Specify the address of the SMTP server used to deliver the alerts. This field is required.

Port Number. Specify the port number used to access the SMTP server. This field is required.

Email From. Specify the address to use as the “From:” in the header of the e-mail message. This field is required.

Authentication. Select this option if the SMTP server to be used requires authentication. When this check box is selected, the User Name and Password fields display.

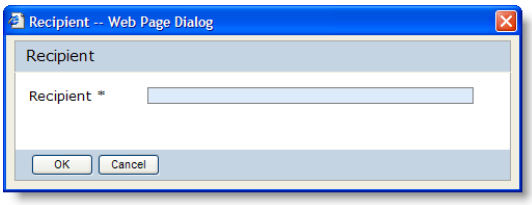
User Name. If authentication is required, specify a user name.

Password. If authentication is required, specify a user password.

Active. Select this option to activate e-mail alerts. No e-mail alerts will be distributed unless this option is selected.

Verify. Click to validate the SMTP configuration settings. The Recipient dialog displays.

Fig. 16.19
Recipient



Enter an e-mail address in the Recipient field and click OK. If the SMTP server is available and can be communicated with, a success message is sent to the specified address and a message appears on the Recipient dialog. If the setup is incorrect—or if the SMTP server cannot be communicated with—an error message displays. You can view logs for e-mail alerts in:

`<QXI webapp>/WEB-INF/logs/alert.log`

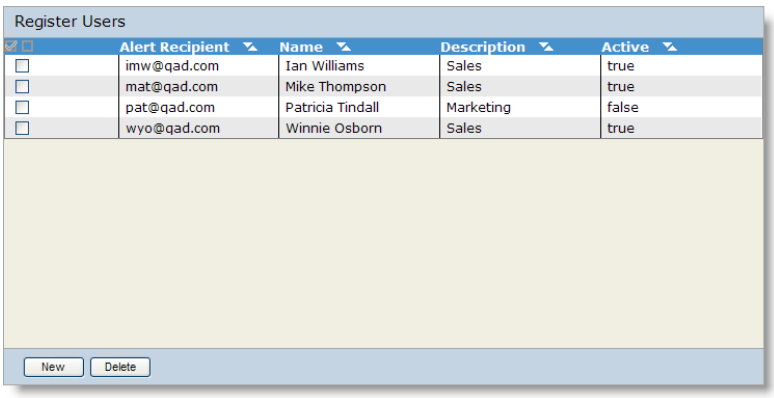
Managing Alert Recipients

Use the Register Users screen to view the e-mail recipients currently defined in QXI. You also can add or delete recipients using this screen.

1 Click the Configuration tab.

2 Choose Email Alerts|Recipients to view the currently defined recipients.

Fig. 16.20
Register Users



Add an Alert Recipient

1 Click New on the Register Users screen to display the Email User Parameters screen.

Fig. 16.21
E-mail User Parameters

2 Enter the following values as required.

Alert Recipient. Specify the e-mail address of the alert recipient. This field is required.

Name. Specify the name of the alert recipient.

Description. Enter a brief description as needed.

Active. Select this option to activate e-mail alerts for the recipient.

3 Click Save to save the new recipient.

Alert Groups

Use the Register Alert Groups screen to view the alert groups currently defined in QXI and the number of members in each group. You also can add or delete alert groups using this screen.

Managing an alert group consists of:

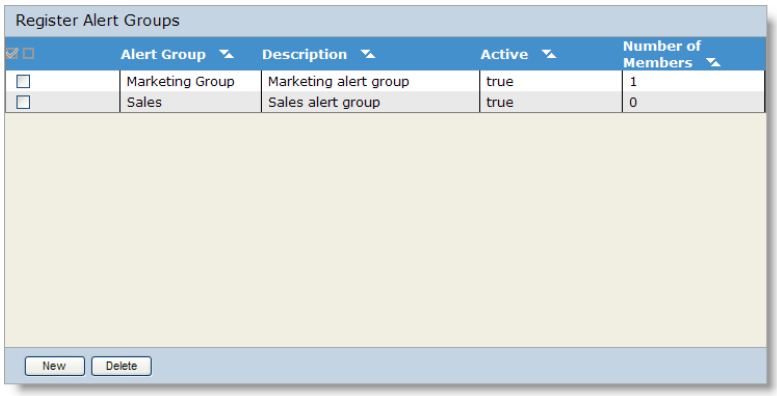
- Adding an alert group
- Registering recipients
- Setting the Catch All Messages option
- Registering receiver, domains, and APIs

Viewing Alert Groups

1 Click the Configuration tab.

2 Choose Email Alerts|Groups to view the currently defined alert groups.

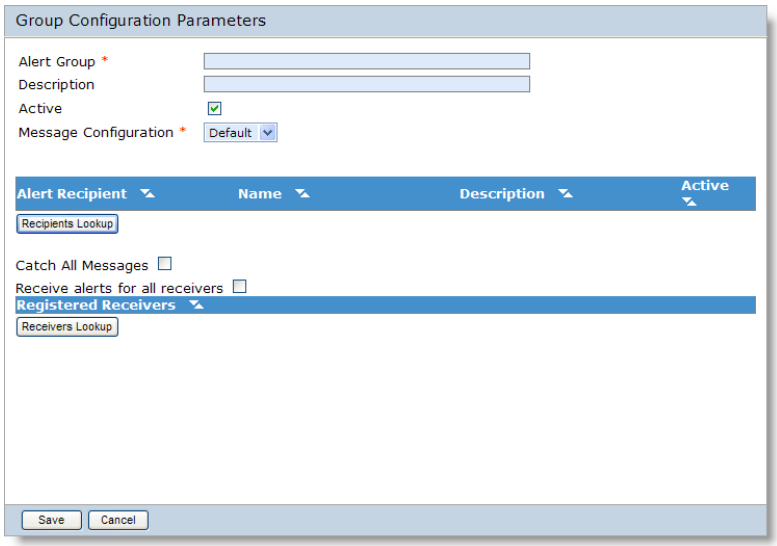
Fig. 16.22
Register Alert Groups



Adding an Alert Group

1 Click New on the Register Alert Groups screen. The Group Configuration Parameters screen displays.

Fig. 16.23
Group Configuration Parameters



2 Enter the following values as required.

Alert Group. Specify a name for the alert group. This field is required.

Description. Enter a brief description as needed.

Active. Select this option to activate e-mail alerts for the alert group.

Message Configuration. Specify whether to use the default message configuration or a custom message configuration.

Default. The message will use the description stored in `qxtendalertconfig.xml` located in

`TOMCAT_HOME/webapps/<QXI webapp>/WEB-INF/emailAlerts/`

Custom. The message will use the subject and body you provide. Selecting this option causes the Email Subject, Email Body, and Attachments fields to display.

E-mail Subject. Enter a brief subject description as needed (for custom message configuration only); for example:

```
A QXtend alert has been triggered.
```

E-mail Body. Enter the text to use to precede the error message and attachments (for custom message configuration only). For example:

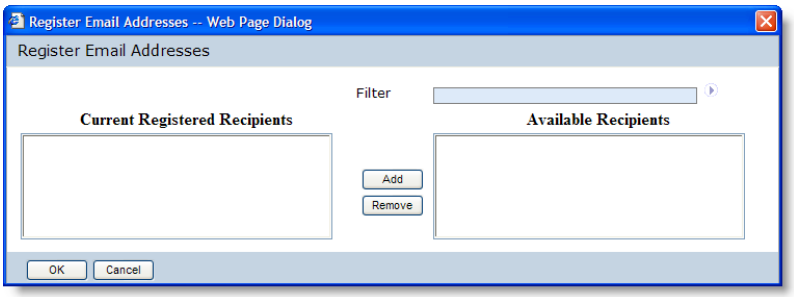
```
There has been a raised alert from QXtend. This event was triggered
automatically and the following message has been recorded:
```

Attachments. Select the Request and/or Response check boxes to include the request and/or response as an attachment in the body of the e-mail alert (for custom message configuration only).

Registering Recipients to a Group

1 Click Recipients Lookup to register the alert recipients to the alert group. The Register Email Addresses lookup displays. Click the arrow next to Filter to populate the Available Recipients list. Use the Add and Remove buttons to update the lists.

Fig. 16.24
Register Email Addresses



2 Click OK to save and exit the Register Email Addresses lookup.

Specifying Which Alerts to Generate

If you want to send to alert group recipients all e-mail alerts that are generated for all receivers, domains, and APIs defined in the system, select the Catch All Messages option. Selecting this option can save you time setting up your e-mail alert groups—receivers, domains, and APIs do not have to registered individually when this option is selected.

If you select this option, specify the severity of the exceptions—ERR, WRN, or MES—that will generate an e-mail alert. Click Save to save your alert group configuration.

Otherwise leave this option clear to specify the receivers, domains, and APIs that you want to generate e-mail alerts.

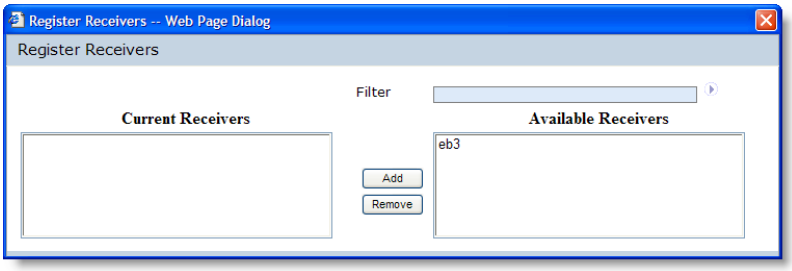
Registering Receivers to a Group

1 Select the Receive alerts for all receivers option to cause e-mail alerts to be generated for all receivers. If you select this option, specify the severity of the exceptions that will generate an e-mail alert, click Save, and then go to step 2 to specify domains.

Otherwise leave this option clear to specify which receivers should generate e-mail alerts in step 2.

2 Click Receivers Lookup to select the receivers to register. The Register Receivers lookup displays. Click the arrow next to Filter to populate the Available Registered Receivers list. Use the Add and Remove buttons to update the lists.

Fig. 16.25
Register Receivers



3 Click OK to save and exit the Register Receivers lookup.

4 For each receiver, indicate the severity of the exceptions that will generate an e-mail alert.

Note The default message types are ERR and WRN.

5 Click Save to refresh the left-hand tree menu.

Registering Domains to a Group/Receiver

1 In the left-hand tree menu click a receiver in a group/receiver hierarchy.

2 Select the Receive alerts for all domains option to receive alerts for all domains for the specified receivers. If you select this option, specify the severity of the exceptions that will generate an e-mail alert, click Save, then go to step 2 to specify APIs.

Otherwise leave this option clear to specify which domains should generate e-mail alerts in step 3.

Note When creating a new domain, the default message types are the same as the associated receiver. So, for example, if the MES check box is not selected for the receiver, the MES check box for domains is disabled. The same applies when registering APIs.

3 Click New to register domains to the group/receiver. The Domain Configuration Parameters dialog displays. Specify a domain name and click Save.

Fig. 16.26
Register Domains

4 Click Save to save the group/receiver/domain configuration.

5 Click Save to refresh the left-hand tree menu.

Register APIs to a Group/Receiver/Domain

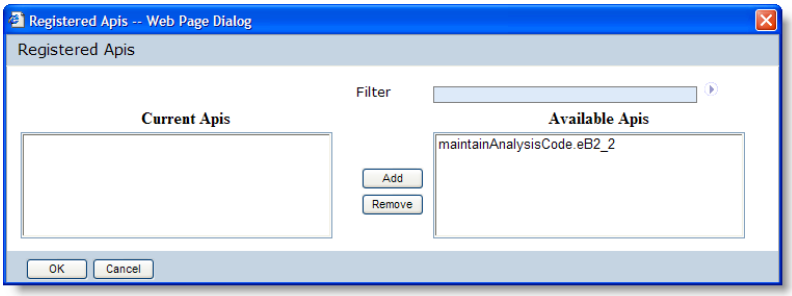
1 In the left-hand tree menu click a domain in a group/receiver/domain hierarchy.

2 Select the Receive alerts for all APIs option to receive alerts for all APIs for the group/receiver/domain. Otherwise leave this option clear to specify which APIs should generate e-mail alerts in step 3.

Note When creating a new API, the default message types are the same as the associated receiver.

3 Click Lookup to register specific APIs to the group/receiver/domain. The Register APIs lookup displays. Click the arrow next to Filter to populate the Available APIs list. Use the Add and Remove buttons to update the lists.

Fig. 16.27
Register APIs



4 Click OK to save and exit the Register APIs lookup.

5 Click Save to save the alert group.

Chapter 17

QXI Connection Pool Manager

The Connection Pool Manager controls the telnet connections between QXtend Inbound (QXI) and your QAD Enterprise Applications sessions.

Introduction

The Connection Pool Manager controls telnet connection pools between QXI and your QAD Enterprise Applications and other application sessions. Each connection pool is identified by the type of user it serves—UI API, JITSAPI, SIAPI, or FinAPI—and by a pool name, the host machine and port, and a system user.

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

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

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

Use the Connection Pool Manager to:

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

Starting the Connection Pool Manager

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

For information on Connection Pool errors, see “Connection Exceptions” on page 356.

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

```
http://<hostname>:<tomcat_port>/<QXI webapp>/
```

You can also access the Connection Pool Manager directly at:

```
http://<hostname>:<tomcat_port>/<QXI webapp>/ConnectionManager.html/
```

Both methods assume Tomcat and the target QAD Enterprise Applications instances are running.

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

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

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

Fig. 17.1
Connection Pool Manager Admin Functions Menu

3 Choose the following menu options to accomplish administrative tasks:

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

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

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

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

Viewing the Connection Pool Log

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

TOMCAT_HOME`/webapps/<`*QXI webapp*`>/WEB-INF/logs`

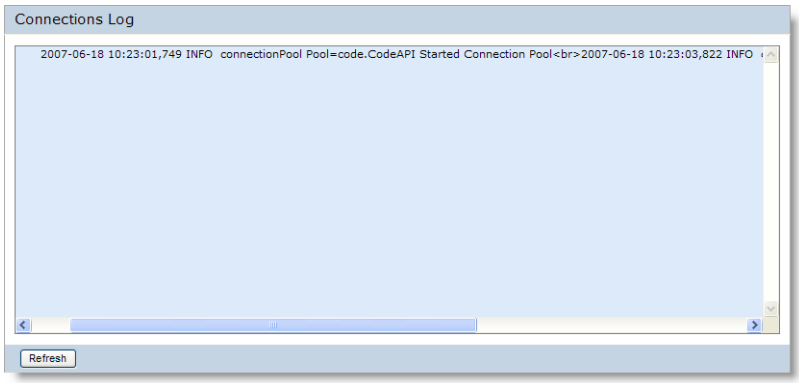
This log tracks the Connection Pool Manager and connection pools over a progression of reporting levels—error, warning, info, and debug.

The error level is controlled by modifying the level value for the logger name connectionPool in `qxtendlogging.xml`. By default, the level is set to Info. See “Log Report Levels” on page 158 for information on reporting levels.

To view the connection pool log in Connection Pool Manager:

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

Fig. 17.2
View Connection Pool Log



2 Scroll down to view the most recent entries.

Configuring Connection Pools

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

Adding a Connection Pool

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

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

TOMCAT_HOME`/webapps/<`*QXI webapp*`>/WEB-INF/conf/defaults`

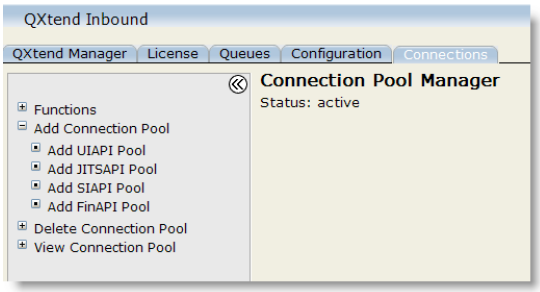
The server-specific version is stored one level up in the `/conf` directory.

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

Adding a UI API Pool

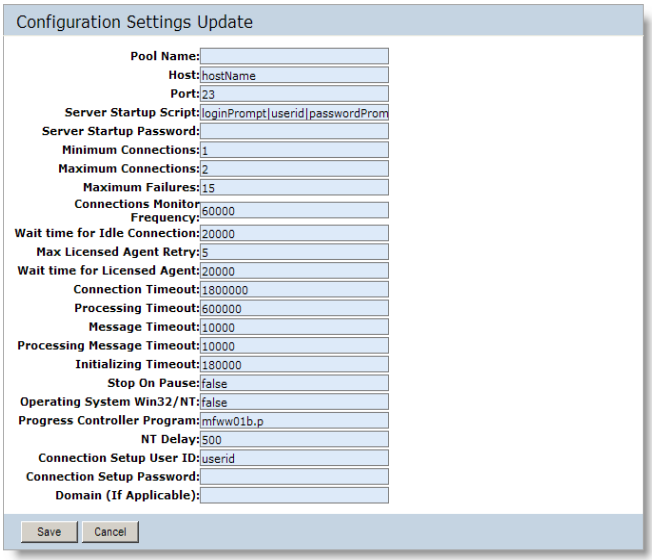
1 In the Connection Pool Manager, choose Add Connection Pool. The available user types display.

Fig. 17.3
Connection Pool Manager



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

Fig. 17.4
Add a UI API Connection



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

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

Port. Enter the port number for the telnet server.

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

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

See “Creating Telnet Log-In Scripts” on page 318.

For Oracle implementations, the qma script must be modified.

See “Progress/Oracle Calling Program” on page 322.

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

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

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

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

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

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

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

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

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

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

See “QAD QXtend Licensing” on page 253 for details on licensing.

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

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

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

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

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

Stop on Pause. For QXI, this should always be set to false. This prevents a transaction from failing when a "Press Spacebar" message is displayed in the target QAD Enterprise Applications session.

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

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

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

Connection Setup User ID. This and the next two entries are the parameters required to connect to the target QAD Enterprise Applications instance. Enter the valid QAD Enterprise Applications user ID, such as qxtend.

Connection Setup Password. Enter the password for the QAD Enterprise Applications user ID (maximum 20 characters). The password is encrypted on entry.

Domain. Enter the valid QAD Enterprise Applications domain if the target instance has domains implemented. Domains were introduced in QAD Enterprise Applications version 2.1.

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

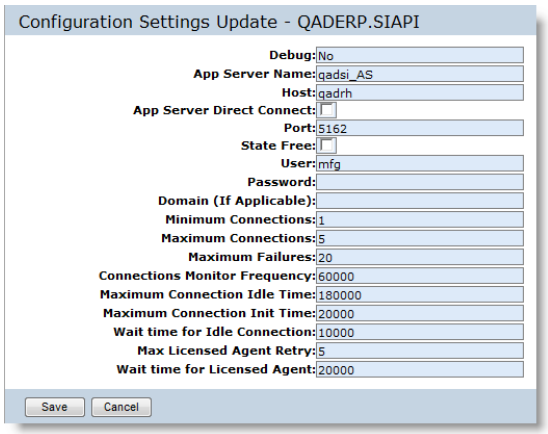
Adding a JITSAPI, SIAPI, or FinAPI Pool

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

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

2 Choose Add JITSAPI Pool or Add SIAPI Pool as needed. The Configuration Settings Update form displays. Use the following field descriptions to create a connection pool configuration.

Fig. 17.5
Add a SIAPI Pool



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

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

App Server Name. Enter the name given to the AppServer in the `ubroker.properties` file. This is configurable and is done by the user when creating the AppServer settings. See Chapter 20, “Configuring the Progress AppServer,” on page 239.

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

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

Yes: QXtend directly connects to the App Server using the port number you provide in the Port field for the AppServer.

No: QXtend first uses the port number you supply in the Port field to connect to the NameServer, which then assigns the AppServer to the connection.

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. In that case, the connection is requested from the NameServer and it assigns the AppServer to the connection.

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. Enter the QAD Enterprise Applications user name.

Password. This is the QAD Enterprise Applications user’s password, encrypted on entry.

Domain. For a SIAPI pool, enter the valid QAD Enterprise Applications domain if the target instance has domains implemented. This field does not apply to JITSAPI pools.

Session Type. This value displays only in the QAD JIT Sequencing API connection pool configuration. See *Installation Guide: QAD JIT Sequencing* for details. Set it to dynamics session type to enable a Progress Dynamics session.

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

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

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

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

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

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

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

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

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

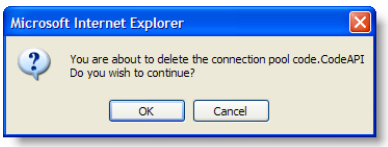
Delete a Connection Pool

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

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

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

Fig. 17.6
Connection Pool Deletion Warning



3 Choose OK to delete the connection pool.

Connection Pool Administration

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

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

Connection Pool States

Each connection has one of the following statuses:

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

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

Busy. The connection is currently executing a user request.

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

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

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

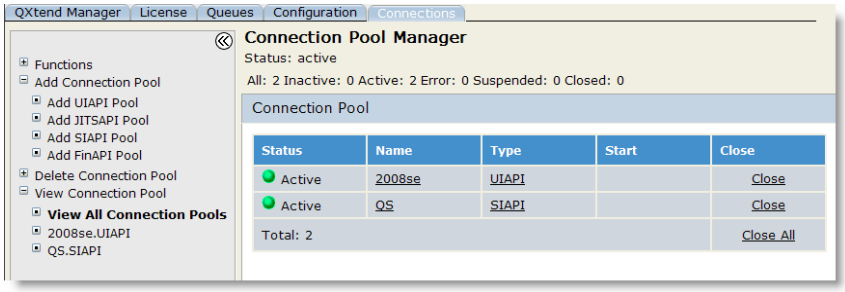
Disconnected. This is a temporary state that occurs when idle connections are closed.

Viewing Connection Pools

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

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

Fig. 17.7
View All Connection Pools



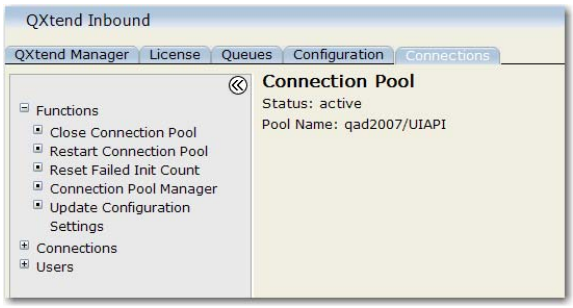
2 Choose an individual connection pool name from the View Connection Pool list to manage a specific pool.

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

Managing a Connection Pool

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

Fig. 17.8
Connection Pool Menus



The following options are available from the Functions menu:

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

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

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

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

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

Choose the Connections menu to view the pool of active connections. See “Connection Pool States” on page 210 for information on the individual states. You can view the connections in the following categories:

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

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

Idle. Displays all idle connections.

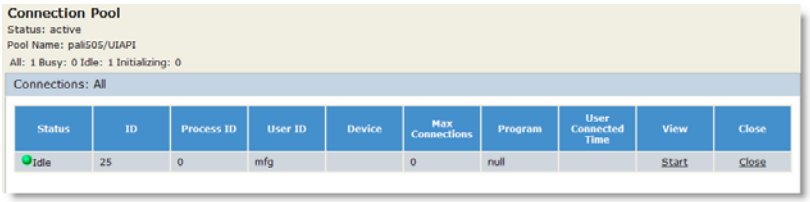
Initializing. Displays initializing connections.

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

Manage User Sessions

Choose Users to view currently logged in users. Choose a user ID to see information related to that user. Figure 17.9 shows an example of a connection pool in which there are no busy sessions (Busy: 0), no idle sessions (Idle: 0), and one user session initializing (Initializing: 1).

Fig. 17.9
Monitoring Users



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

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

Debugging Your QDocs

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, choose the Stop link, which displays next to the Start link.

Important Since telnet windows for connections use system resources, you should close any telnet windows not in use.

Viewing Multiple Telnet Windows

Each `<ConnectionPool>` section has its own base port. By default the port value is set to 15000 for every connection pool.

```
<snoopingPort label="Snooping Base Port" uiconfig="false" value="15000"/>
```

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.

Connection Pool Manager Scripts

This script makes use of GNU sed and GNU wget so that it works on most flavors of UNIX, including HP-UX and AIX. So before running this script, please install the GNU sed and GNU wget, which are freeware downloads available from http://ftp.gnu.org/gnu/. Once both sed are gnu

are installed, copy the executables (sed and wget) from the directory you installed (default `/usr/local/bin`) to this scripts directory to make sure the GNU versions of the programs are running and not the native OS ones.

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:

```
conn-control.sh -Operation [Pool] [Type]
```

Where *`Pool`* is the connection pool name and *`Type`* is the connection pool type, which is mandatory when *`Pool`* is used.

Operation	Description
h	Print the script help
q	Query the connection pool status
r	Restart connection pools
s	Start connection pools
x	Stop connection pools

Example Use the following command to stop the connection pool named QADQXO with type SIAPI:

```
./conn-control.sh -x QADQXO SIAPI
```

Chapter 18

QGen

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

Introduction

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

In general, QGen is used to map custom programs or QAD Enterprise Applications programs that do not have released QDoc schemas and events files. You can also use QGen to modify the standard QAD QDoc schemas and events files by loading the standard `.dat` file and running through the menu-level program again. See “Load” on page 228 for details.

In addition, you can use the native API framework available from QGen to modify QAD Enterprise Applications character programs; for details see “Generating Native APIs” on page 217.

The standard QGen process is:

1. Launch an QAD Enterprise Applications QGen session.
2. Map a program:
   a. Open the QAD Enterprise Applications program.
   b. Launch QGen.
   c. Map individual fields and iterations.
   d. Save your work.
3. Generate the QDoc schemas and events file.

In addition, QGen enables you to:

- Modify individual field information after mapping is complete.
- Save partially mapped programs and reopen them later.
- Discard program maps and restart the mapping process.
- Add schema that was changed with one of the following:
  - .NET using configured/simplified screens
  - ICT (Integrated Customization Toolkit)
  - Native API programs
  - The SIAPI function

Program Structure and Terminology

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

As you proceed through an QAD Enterprise Applications program, you encounter the following types of fields:

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

The primary key is used to determine which fields are populated in the response document; only the fields defined as primary keys appear in the response schema.

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

Mandatory. Yes indicates the field must have a value.

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

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

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

QGen Files

The QGen tool is installed as part of the QAD QXtend installation. For details, refer to *Installation Guide: QAD QXtend*. When the installation is complete, the QGen program files are installed in the `<QXtend adapter>/tools` directory.

Generating Native APIs

By using the native API framework you can run standard QAD Enterprise Applications character programs on an AppServer and receive input from a dataset rather than from the user interface. In addition, the framework allows you to run character programs that have been customized using the QAD Integrated Customization Toolkit (ICT) or the .NET UI.

Note For details on customizing functions using the .NET UI, refer to Chapter 8, “Configurable Screens,” in *User Guide: QAD User Interfaces*.

The .NET UI allows you to configure a screen to suit your requirements by disabling fields, setting default values, and so. If the modified function is supported by a native API, that function can be used in native API mode with the identical business logic. For example, if a QDoc contains a value for a field that has been disabled in the customized function, the value for the field will not be set in the API.

Note For a list of functions that are supported by native APIs, see “Native APIs” on page 240.

Since configurable screens are configured by role, the system uses the role associated with the user ID specified within the QDoc.

.NET Customization

The .NET UI supports the following customizations:

- Disabled fields. These are supported by ensuring that the field in the request is always set to the unknown value so that the API determines what the value will be.
- Default values. These values are supported by ensuring that in the case of an Add operation, if the request field is unknown, the default value is put in its place.
- Required fields. These fields are supported by ensuring that the field in the request, if it is a character data type, is not set to blank. A post-processing step ensures that after the API has executed, required fields in the database are also not set to blank.
- Validation. Field validation is supported by allowing the customization class to validate a field and its value however it chooses.
- User fields. These fields are in the same database table as the other fields in the request, but are not part of the static API dataset. User fields are supported by copying the value from the request to the database as a post-processing step.

  Note The post-processing step is an internal process rather than an operation that can be physically performed. This means there may be some limitations to your customizations.

  Example You have a customization using ICT that updates pt__chr01 in frame A and runs a program when frame A finishes that uses the value the user entered in pt__chr01 to set another value. The additional fields are updated as a post-processing step outside of the API execution, and in the above scenario the value of pt__chr01 may not be set when the custom program is executed.
- New fields. These fields are supported by allowing the customization class to get and set the value of the new fields.

Important For the API to work correctly with .NET customizations—as well as ICT customizations—the .NET UI and/or ICT programs must be specified in the PROPATH of the session running the API.

ICT Customization

The ICT supports the same customizations as .NET except required fields and custom navigation. In addition it supports the following customizations:

- Shadow tables
- Program modification using tags

Note Any external program that is executed by the ICT process must also be modified to work in API mode.

Some ICT customizations are based on ENTRY, LEAVE, and GO triggers. Since these triggers never occur in API mode, customizations involving custom program execution is performed as part of the post-process step. One problem that may arise is that these triggers can be frame and field based, and if the field appears in more than one frame then it is possible to run a different program depending on the frame it is in. This situation will have to be managed in the custom program itself.

Native API Process Flow

Creating and using native APIs consists of several steps:

1. Customize your character program(s) as required using either .NET UI or ICT.
2. Configure the `controllers.xml` and `customizations.xml` files as required. See the “Setting Up Customization” on page 219.
3. If additional fields have been added to a function, a new schema *must* be generated for the role. Upload the new schema files to the QXI server or to a file.
4. Specify the .NET UI and/or ICT programs in the PROPATH of the session running the API.

Setting Up Customization

If you are changing the response schema file name, you must edit the `controllers.xml` file. Also, if you do not want to have customizations turned on, you must edit the `customization.xml` file. Both of these files are located in the `qxtendadapter/config` folder.

controllers.xml

The `controllers.xml` file lists all available native APIs. Each API to be used must specify the name of the API (for example, `maintainMasterComment`), the name of the customization class, the name of the request schema and response schema (both optional), and API program name, as in the following example:

```
<?xml version="1.0"?>
<controllers>
    <controller>
        <apiName>maintainMasterComment</apiName>
        <className>com.qad.mfgpro.api.MasterCommentController
        </className>
        <requestSchema>maintainMasterComment-ERP3_1.xsd</requestSchema>
        <responseSchema>maintainMasterCommentResponse-ERP3_1.xsd</responseSchema>
        <apiProgram>gpcmmt.p</apiProgram>
   </controller>
   <controller>
        <apiName>maintainItemSitePlanning</apiName>
        <className>com.qad.mfgpro.api.ItemSitePlanningController
        </className>
        <requestSchema>maintainItemSitePlanning-ERP3_3.xsd</requestSchema>
        <responseSchema>maintainItemSitePlanningResponse-ERP3_3.xsd</responseSchema>
        <apiProgram>pppsmt02.p</apiProgram>
   </controller>
    ...
    ...
```

```
</controllers>
```

customizations.xml

Each type of customization must be defined in `customizations.xml`, as in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<customizations>
	<customization>
		<name>.Net UI Customizations</name>
		<class>com.qad.mfgpro.api.UICustomization</class>
	</customization>
	<customization>
		<name>ICT Customizations</name>
		<class>com.qad.mfgpro.api.ICTCustomization</class>
	</customization>
</customizations>
```

where `name` is the name of the customization and `class` is the name of the class for the customization. If customizations will not be used, these can be commented out.

Note Commenting out ICT customizations that are not required will marginally improve system performance.

Starting QGen

To start QGen, run script `<qxtend adapter>/scripts/client.qgen` on Unix or `clientqgen.bat` on Windows.

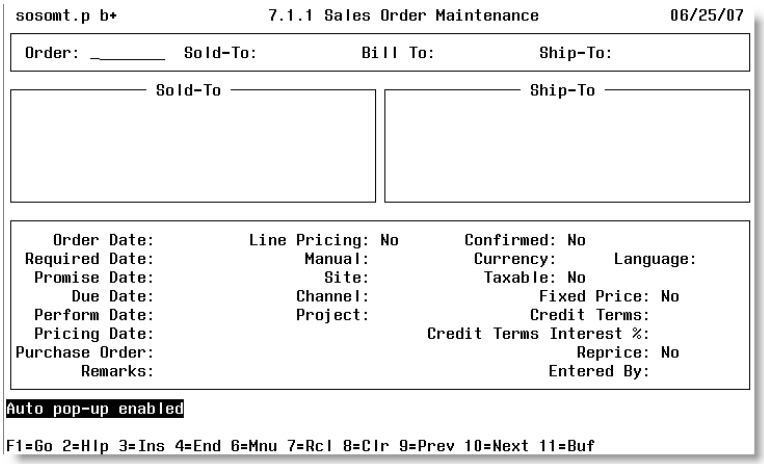
This starts an QAD Enterprise Applications session with QGen running in the background.

Important Use this QAD Enterprise Applications session only to generate QDocs, not to enter data.

Mapping a Program for Regular UIAPI Interface Programs

1. Navigate to the program in QAD Enterprise Applications that you want to generate QDocs for.
2. In the first field, press Ctrl+W. A message, Auto pop-up enabled, displays.

Fig. 18.1
Pressing Ctrl+W in a QGen Session

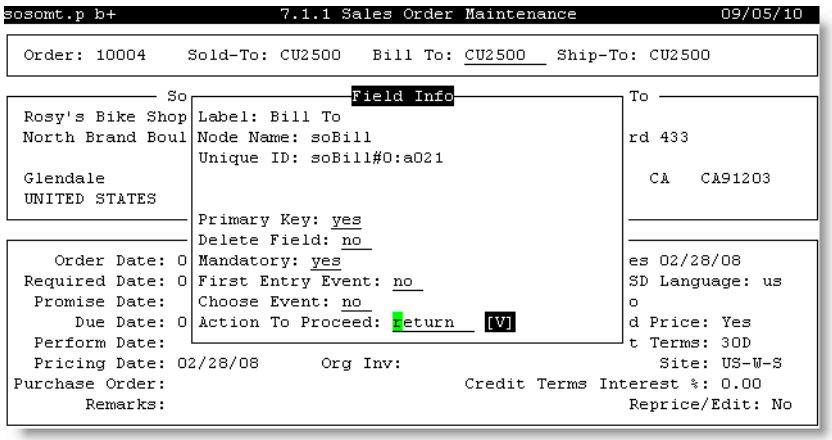


3 Enter data in the first field or select an existing record to edit and press Enter to move to the next field.

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

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

Fig. 18.2
Field Info Pop-Up



Label. The label value is the field label that displays on the QAD Enterprise Applications screen. This is not available for editing.

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

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

Primary Key. A Yes in this field designates this field as one of the primary keys of the parent or child records. For more on iterations, see “Map Iterations” on page 223.

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

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

Mandatory. Yes indicates the field must have a value.

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

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

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

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

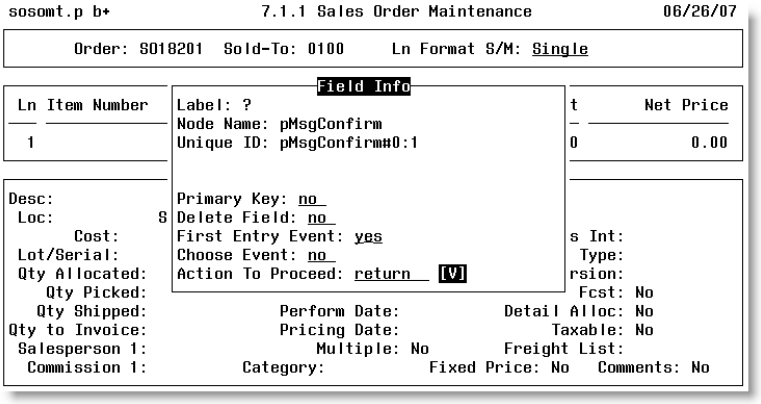
5 If First Entry Event for the field is set to Yes, the First Entry Event pop-up displays. Otherwise, you move to the next field.

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

Map First Entry Events

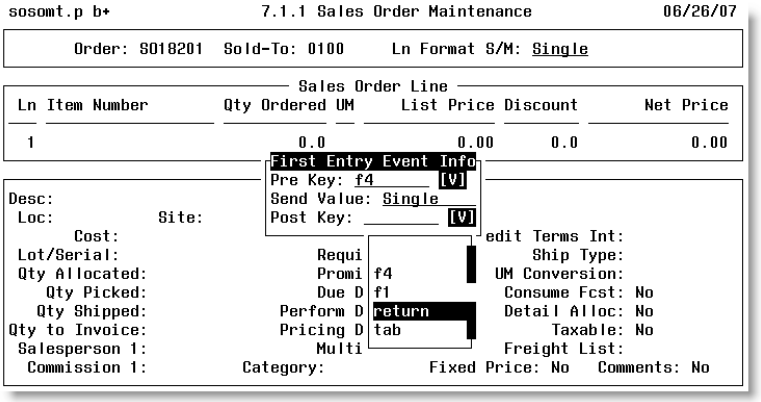
A first entry event occurs at a branching point in a program, usually prior to starting an iteration. Typically the field is accessed by pressing F4 from the first field of an iteration.

Fig. 18.3
First Entry Event Field Definition



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

Fig. 18.4
First Entry Event Info Pop-Up



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

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

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

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

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

Map Iterations

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

To accomplish this, you navigate through the first field of the frame twice. The first time, enter data normally as described in “Mapping a Program for Regular UIAPI Interface Programs” on page 220. The second time through, make sure to return to the first field in the frame. QGen automatically recognizes the iteration.

One common iteration type is the Comments iteration. This does not need to be mapped because it is defined in the `commonTypes.dat` file. See “Starting QGen” on page 220.

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

False Iterations

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

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

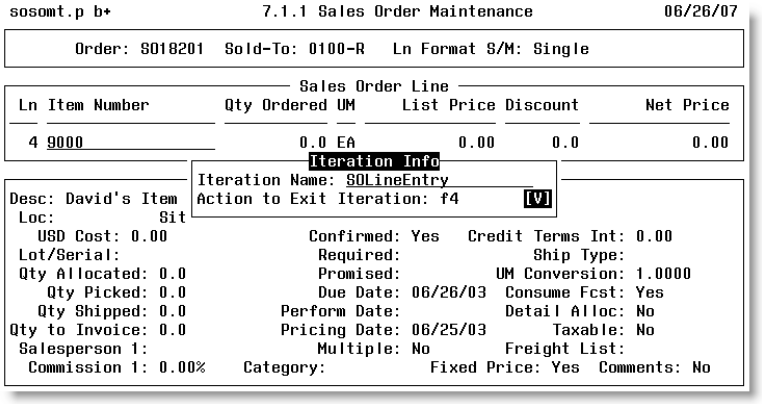
Hidden Iterations

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

Mapping an Iteration

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

Fig. 18.5
Iteration Info Pop-Up



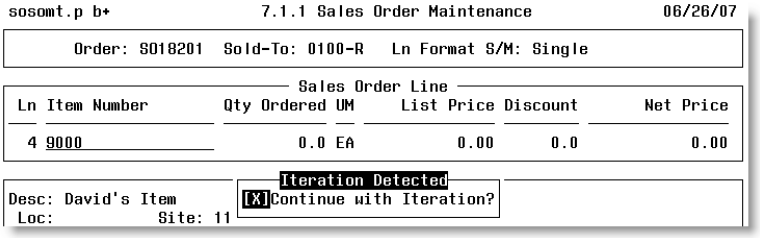
Enter the following data:

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

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

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

Fig. 18.6
Continue With Iteration Query



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

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

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

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

```
Please exit the iteration.
```

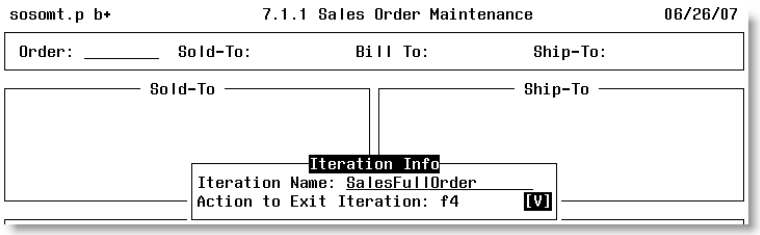
Press spacebar to select the message. Press Go to exit the iteration message. Move on to step 6.

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

Note If the iteration is exited in a different way than you originally specified (Action to Exit Iteration was set to F4 but should be F4:F4), you can change it in Update Mode. See "Change Mode to Update" on page 228.

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

Fig. 18.7
Complete Program Iteration



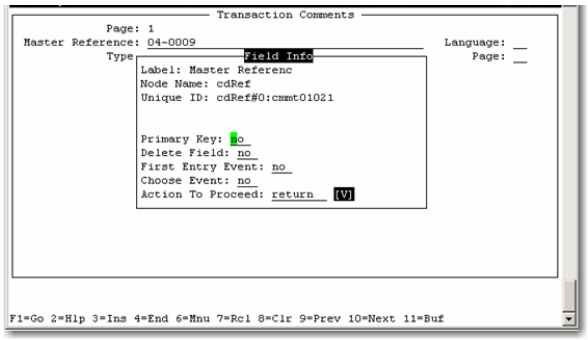
To finish the mapping process, press Ctrl+W to disable the pop-ups.

Map Comments

When you enter the Transaction Comments section, use the following process to map the screen:

1 Press Enter in the Master Reference field.

Fig. 18.8
Transaction Comments



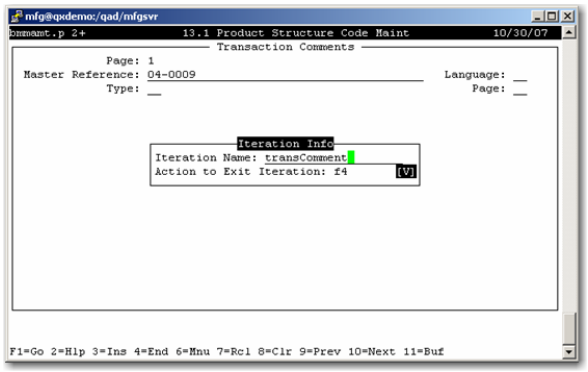
2 Continue to map the remaining fields.

3 When you are on the first line of the Comments, press F4 to return.

4 Press Enter on the Master Reference field again and QGen will detect this as an iteration.

5 Name the iteration transComment.

Fig. 18.9
Iteration Name



6 Finish the iteration and complete the mapping process.

7 Save, generate, and load the QDoc schema into QXI.

8 Use the Test Harness to create an empty QDoc.

9 Edit the QDoc with data to submit. Ensure to include data in the comments section.

10 Submit the QDoc using the QXtend Test Harness or the queue.

Running QGen Options

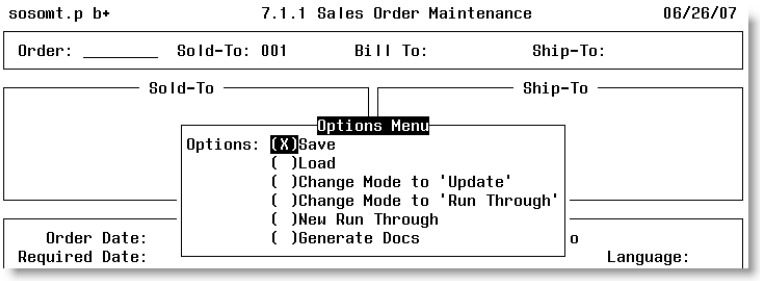
At any time during an QAD Enterprise Applications QGen session, press Ctrl+O to open the QGen Options menu. The options are:

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

Save

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

Fig. 18.10
QGen Options Menu on Save

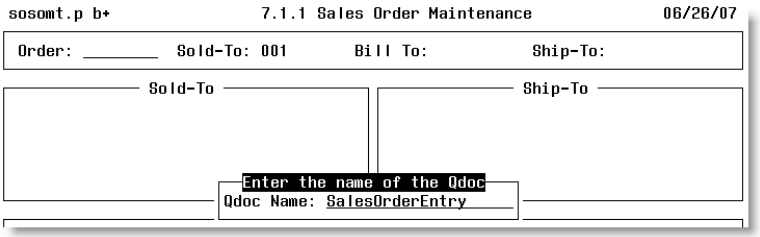


1 Press spacebar to select Save if it is not already selected.

2 Press Go to save.

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

Fig. 18.11
QDoc Save Frame



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

Load

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

1. Arrow to Load and press spacebar.
2. Press Go to load the `.dat` file.
3. You are prompted to enter the name of the file; for example, `SalesOrderEntry`. The QDoc must exist in the correct location and the capitalization must be correct. The `.dat` extension is not required.
4. Press Go to load the document into the current QGen session. The message `Finished loading data` displays.
5. Open the QAD Enterprise Applications program the `.dat` file was created from.
6. Tab through the program fields. Each mapped field displays the message:

   `<fieldname> found in table.`

   The Field Info pop-up displays automatically at the first unmapped field. See “Mapping a Program for Regular UIAPI Interface Programs” on page 220.
7. Select the Save option to save any changes (Ctrl+O, Save).

Change Mode to Update

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

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

Change Mode to Run Through

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

1. Arrow to Change Mode to Run Through and press spacebar.
2. Press Go to switch to Run Through mode.
3. Map programs as described in “Mapping a Program for Regular UIAPI Interface Programs” on page 220.

4 Select the Save option to save your work.

New Run Through

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

1 Arrow to New Run Through and press spacebar.

2 Press Go to start a new run through.

3 You are asked, “Are you sure you wish to reset?” Press spacebar to confirm and press Go. The session information is deleted.

4 Open the QAD Enterprise Applications program you want to remap and press Ctrl+W to start the mapping process.

Generate Docs

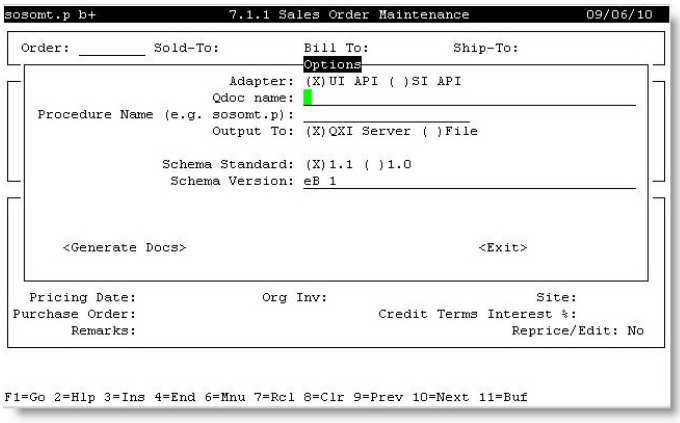
This option generates a QDoc schema and events file from the program information that has been mapped and saved in a `.dat` file. You can specify which standard to use for the generated schema. The available options change according to which schema standard is specified.

Note If you intend to generate native APIs for configurable screens that have been customized using the .NET UI or ICT, see “Generating Native APIs” on page 217.

1 Arrow to Generate Docs and press spacebar.

2 Press Go to start the generation. The Generate Options frame displays.

Fig. 18.12
Generate QDocs Pop-up



3 Fill in the information using the field descriptions below:

Adapter. Select the adapter you want to use.

UI API: Select this option to use the UI API.

SI API: Select this option to use the SI API to generate service interface APIs for configurable screens. Only QAD 2009 SE, QAD 2009 EE and later versions have SI API.

QDoc name. Enter the QDoc name as saved in the QGen `.dat` file; for example, `SalesOrderEntry`.

Note If you select the SI API option and enter an invalid QDoc—that is, a non-SI API QDoc that has not been defined in the `controllers.xml` file—the system displays an error message.

Procedure Name. (1.1 schema standard and UI API only). Enter the name of the QAD Enterprise Applications program name plus a `.p` extension; for example, `sosomt.p`.

User. (1.1 schema standard and SI API only). Enter the name of the user for whom the customization is intended. If the field is left blank, the customization applies to all configurable screens that are associated with all roles.

4 Do either of the following:

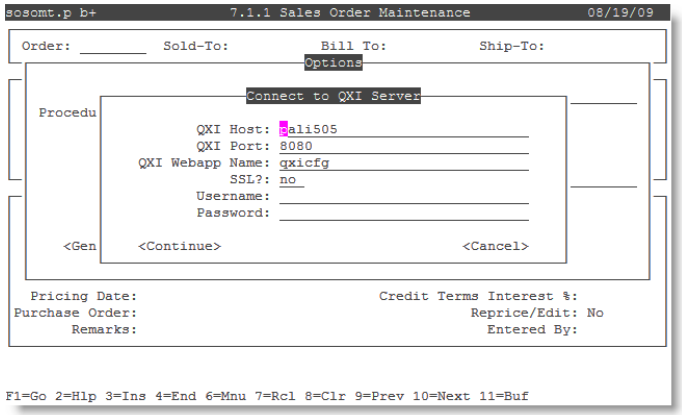
- To generate the schema and events to a file, select the File option. You must specify the results directory in which to store the generated schema and events files, the schema standard, and the schema version. Then select Generate QDocs to generate the files.
- To publish the APIs directly to the QXI server, select the QXI Server option, then continue to step 5.

5 Specify the standard to use for the generated schema. The 1.1 specification is the default standard. (This option does not display if the SI API option is selected.)

6 Specify the schema version to use; for example, eB2_3 for the third version of the file in the eB2 release. The default value is eB_1.

7 Select Generate QDocs. The Connect to QXI Server screen displays.

Fig. 18.13
Connect to QXI Server



8 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.

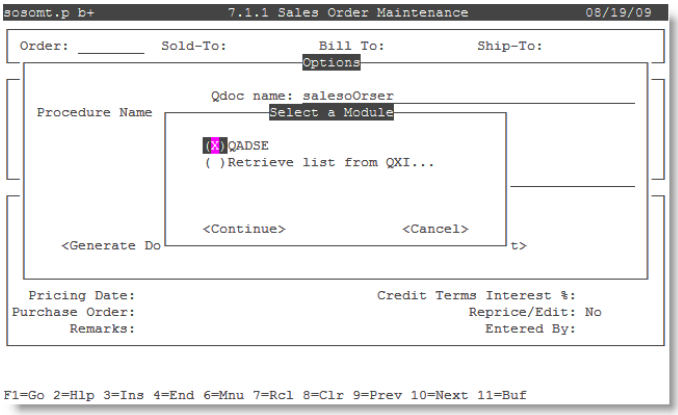
Note If SSL is selected, then QXI Host must include domain name of the host.

Username. Specify the username login for the QXI server.

Password. Specify the password for the user.

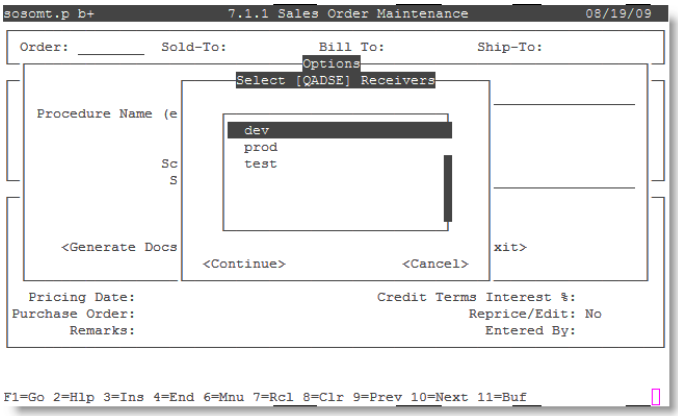
9 Select Continue. The Select a Module screen displays.

Fig. 18.14
Select a Module



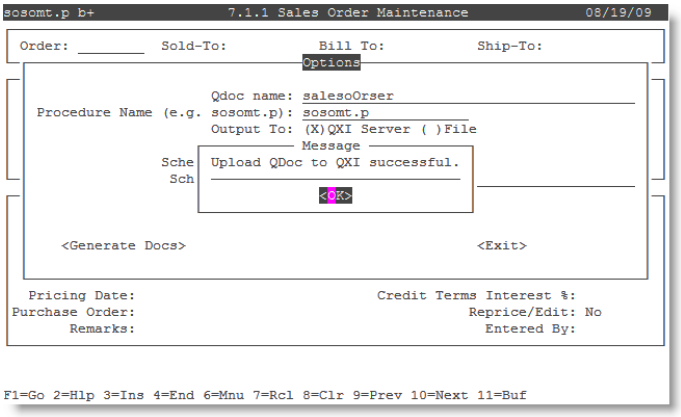
10 Select the module to which you want to send the QDocs and then select Continue. The Select Receivers screen displays.

Fig. 18.15
Select Receivers



11 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.

Fig. 18.16
Upload Successful



12 Select OK to exit.

Troubleshooting QGen

This section documents some possible error conditions that can occur while using QGen and offers suggested resolutions.

Problem: The Automatic pop-up is enabled, but a pop-up is not displayed on exiting the field.

Solution 1: First ensure that the mode is Run Through and not Update.

Solution 2: If QGen is in Run Through mode, check if you are in the message area. This is the area at the bottom of the screen where messages pop up—the cursor is at a question requiring a Yes/No answer. If so, the Field Info pop-up does not display. Continue past the field in the message area. When the next field is reached, QGen displays a pop-up containing the field information for the field in the message area before displaying a pop-up containing information for the currently selected field.

Solution 3: If you are not in the message area, disable the automatic pop-up using Ctrl+W, then tab back to the field and use Ctrl+F to manually map the field.

Error message: Unable to create directory nnnn

This error occurs when the current user does not have permissions to create the specified directory.

Error message: The file “qdoc.dat” has no root node name

The first field in a program was not mapped as an iteration. When mapping a program, the first field in the program must be selected again and an iteration name entered. Exit the Generate Docs menu. Tab to the first field in the program and press Ctrl+F to map it.

Chapter 19

QXtend Inbound Pre- and Postprocessors

This section describes how to use the QAD QXtend Inbound (QXI) pre- and postprocessors to validate requests or verify complete QDocs.

Introduction

The QXI preprocessor lets you call custom programs to validate or modify inbound QDocs prior to writing data to QAD Enterprise Applications. The preprocessor programs can eliminate QAD Enterprise Applications errors, filter required records, eliminate records based on unique sets of criteria, or add or remove data as required.

Important QAD discourages the practise of performing database updates using a preprocessor program due to the significant effects this can have upon system performance. If you decide to perform an update, the update will not be within the same transaction as the QDoc being processed. Consequently if the QDoc fails, the preprocessing transaction will be complete, but the QDoc processing will be incomplete. This situation can be avoided by setting the `scopeTransaction` attribute in the QDoc to True in order to scope the processing of the QDoc to a single transaction, but doing so can result in the performance issues mentioned.

The postprocessor is similar but is used to validate and modify the response document contents. One example is obtaining updated data from QAD Enterprise Applications, such as line item prices that changed after the initial data update from the QDoc request.

The method for implementing and using the QXtend pre- and postprocessors is:

- Enable pre- and postprocessing capability.
- Update the events files to reference the pre- or postprocessing program.
- Create the custom pre- or postprocessor program.

Enabling Pre- and Postprocessing in QXtend

To initiate the ability to run pre- and postprocessing in QXtend, edit the `qxtendconfig.xml` file. If you have configured QXtend to enable field values in response documents, you should have already made this change. See “Include QAD Enterprise Applications Field Values” on page 156.

1. Open *`TOMCAT_HOME`*`/webapps/<`*`QXI webapp`*`>/WEB-INF/conf/qxtendconfig.xml.`
2. Locate the `messageServletURL` attribute within the `<general-config>` node.
3. Enter the correct host and port values to communicate with the QXtend server.

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

4. Save and close the file.

Updating Events Files

The events files in QXI are `.xml` files containing program-specific keystroke and navigation. Events files for each QDoc-supported program are shipped with the product or are generated by the QGen product. You can maintain multiple copies of the events files for each QAD Enterprise Applications program to support either different paths through a program, or to test incremental performance improvements in update paths.

These files must first be copied to the custom events directory; then a new event and the new program name must be added.

1 Copy the events file you want to modify to the `custom` directory. The following example is for sales orders. Copy from:

TOMCAT_HOME/webapps/<*QXI webapp*>/WEB-INF/events/eB2/sosomt-eB2_1.xml

to:

TOMCAT_HOME/webapps/<*QXI webapp*>/WEB-INF/events/eB2/custom/sosomte-B2_1.xml

2 Edit the events schema file to add a new IterationEvent. The event must have an attribute of either `preprocess` or `postprocess`. This attribute defines the custom program to call when the iteration event is encountered. Use the following code fragment showing a preprocessing program as a model:

```
<field uid="soNbr#0:a02" name="so_nbr" inheritevents="true">
    <IterationEvent iterationname="salesOrder" exititeration="f4" preprocess=
    "xxMyPreProg.p"/>
</field>
```

This sample runs `xxMyPreProg.p` before processing the QDoc.

An equivalent postprocessing program call would appear as follows:

```
<field uid="soNbr#0:a02" name="so_nbr" inheritevents="true">
    <IterationEvent iterationname="salesOrder" exititeration="f4" postprocess=
    "xxMyPostProg.p"/>
</field>
```

Creating Custom Pre- and Postprocessing Programs

The custom pre- and postprocessing programs must be written in the Progress 4GL. The program you create is called from a Progress program named `mfww01b.p`. This Progress program has procedures that your program can call to limit your programming task to the validations you require.

Program Structure

The program structure is as follows:

```
DEFINE INPUT PARAMETER pQDoc AS HANDLE NO-UNDO.
DEFINE INPUT PARAMETER pMessageLogger AS HANDLE NO-UNDO.
    /* Custom validation code goes here; */
    /* Log any warnings or errors.*/
RETURN ERROR.
```

Program Naming and Location

Store the file in the standard custom code directories defined for QAD Enterprise Applications:

MFGPROInstallDir/us/xx/xxsosomt.p

This path should be included in your PROPATH, and should reflect the actual installation language if it is not US. The standard QAD naming convention for custom programs is to prefix the file name with `xx`. You may need to adjust the file name if customized programs already exist for the program—for example, change `xxsosomt.p` to `xxsosoqx.p` if `xxsosomt.p` already exists.

QDoc Iterations

Iterations are steps through the data nodes, such as sales order lines, of a given inbound QDoc. These nodes and their contents are passed into the Progress calling program as an X-DOCUMENT. The input parameter `pQDoc` is a handle to the Progress X-DOCUMENT. The custom program can access the nodes within this X-DOCUMENT and carry out any validation required.

Note The custom program is not given access to all of the QDoc, only the current iteration node.

For example, a preprocessing program for SOLine iterates on each sales order line. In the sample portion of the QDoc below, the bold salesOrderLineDetail iteration is the current iteration when the event triggers for sales order line 1. When the event triggers for sales order line 2, the italic salesOrderLineDetail iteration is the current iteration. The following example has been edited for simplicity—lines where three periods appear (...) show where lines have been removed.

```
        ...
        <salesOrder>
            <soNbr>dql0009</soNbr>
            <soCust>1012000</soCust>
            <soConsignment>no</soConsignment>
            <salesOrderLineDetail>
                <line>1</line>
                <sodPart>ALX001</sodPart>
                <sodtaxable1>no</sodtaxable1>
                <sodcmmts>no</sodcmmts>
            </salesOrderLineDetail>
            <salesOrderLineDetail>
                <line>2</line>
                <sodPart>ALX002</sodPart>
                <sodtaxable1>no</sodtaxable1>
                <sodcmmts>no</sodcmmts>
            </salesOrderLineDetail>
        </salesOrder>
    </maintainSalesOrder>
```

Warnings and Errors

If an error occurs, the custom program must log the error using the logError procedure in `mfww01b.p`, then return `Error` to stop processing the QDoc. If an `Error` is returned, the next QDoc is selected and preprocessing is run on it. If there is a warning or the processing is successful, your program should return nothing.

You can log all warnings and errors that arise during pre- or postprocessing by calling the `logWarning` or `logError` subprocedures in `mfww01b.p`.

To log a warning message:

```
run logWarning in pMessageLogger (input "Warning: sales order SO9945 not confirmed.").
```

To log an error message:

```
run logError in pMessageLogger (input "Error: sales order 9956 incorrect so_nbr
format.").
```

Any messages logged using these methods appear in the response QDoc.

Note Logging an error does not stop the QDoc from processing; you must return `Error` to stop it from processing.

Modify the QDoc Iteration Node

If your custom program modifies the QDoc iteration node that is passed down, the program must run the `updateXMLDocument` procedure provided in `mfww01b.p`. The `updateXMLDocument` procedure requires a handle to the new updated X-DOCUMENT as a parameter. Here is an example of how you would call the `updateXMLDocument` procedure:

```
run updateXMLDocument in pMessageLogger (pQDoc).
```

Calling `updateXMLDocument` as illustrated here ensures the iteration node that was sent to the preprocess program is updated with `pQDoc`.

Chapter 20

Configuring the Progress AppServer

This section provides information about the setup required for using service interface and QAD JIT Sequencing APIs with QAD QXtend Inbound (QXI).

Introduction

QXI relies on the Progress open client and AppServer for the available service interface APIs in order to process synchronous API requests.

Native APIs

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.

Note Native APIs are available in the QAD QXtend 1.6 version; they are not available in earlier versions. For more information about native APIs—and using the native API framework—see “Generating Native APIs” on page 217.

In addition, using native APIs allows responses to be customized. Whereas QXI is responsible for populating the response when the UI API adapter is used—in addition to dictating the format of the response—using native APIs permits the target application to determine the appearance of the response.

Native APIs are provided for:

- Routing Maintenance
- Master Comment Maintenance
- Product Structure Maintenance
- Item Cost Maintenance
- Item Data Maintenance
- Item Inventory Data Maintenance
- Item Planning Maintenance
- Item Site Cost Maintenance
- Item Site Inventory Data Maintenance
- Item Site Planning Maintenance
- Item Copy
- Item Maintenance
- Available To Promise (ATP) Query
- Pricing What-If Query

The response schema definitions are located in the same directory as the native API code: `<QXtend Inbound adapter>/schemas`

In maintenance APIs, the response QDoc can contain any field or table that is within the request QDoc.

By default, all fields in the request are provided in the response.

In query APIs, the response by default will contain every available field. If the request XML does not contain a node that is available for the request, it will accept the default—it will not return a blank. The presence of an empty node can be explicitly stated by setting the node value to `xsi:nil="true"`.

Unlike the UI API adapter, native APIs validate the operation, so if the request tries to add a record that already exists in the database, it returns an error—the same applies for modify and remove operations.

Transaction Comments in Native APIs

Any transaction comments in native APIs will be 0-based, so a cmtSeq value of 0 will show as Page 1 in QAD EA.

For the maintainRouting API, for the transaction comments part, the API will automatically increase `<cmtSeq>` and `<masterCmtSeq>` by 1, then process the request.

For example, the following content in the request xml will create a comment with page 1, and the content will be copied from master comment 1.

```
<ttRoutingComment>
<operation>A</operation>
<roRouting>Q012</roRouting>
<roOp>2</roOp>
<roStart>2009-01-31</roStart>

<cmtSeq>0</cmtSeq>
<cmtRef>Q001 comm</cmtRef>
<cmtLang>us</cmtLang>
<cmtType>AB</cmtType>
<cmtPrint></cmtPrint>
<masterCmtSeq>0</masterCmtSeq>
</ttRoutingComment>
```

Windows Setup

If the AppServers are running on Windows, use the Progress Explorer tool to configure the NameServer and the AppServer instances. See “Progress NameServer Setup” on page 242 and “Progress AppServer Setup” on page 243 for details on configuring the NameServer and AppServer instance properties.

The AdminServer, NameServer, and AppServer can be started and stopped using the Progress Explorer tool; they can also be set to start automatically. All administration for the AppServer and the AdminServer can be done using the Progress Explorer tool.

Non-Windows OS Setup

Configure the AppServer manually by editing scripts in the Progress install directory. The information in this section does not detail every possible setup scenario. For details on AdminServer, NameServer, and AppServer configuration, see the Progress documentation.

Progress AdminServer Details

The Progress AdminServer is started using the `proadsv` script. The AdminServer uses the `<DLC>/AdminServerPlugins.dat` file to start the AdminServer by default. Default ports used for the AdminServer are:

- AdminServer Port: 20931
- AdminServer to Database Communication Port: 7834

These ports are the Progress defaults and can be set by passing startup parameters. See “Parameter File Setup” on page 242.

The port the AdminServer is running on is used when starting and stopping NameServer and AppServer connections. Consideration must be given to which ports are going to be used for implementation if the default ports are not available or acceptable.

Progress NameServer Setup

The NameServers are configured in the `ubroker.properties` file. This file resides in the *`DLC`*`/properties` directory and contains all the configuration settings for all the NameServers and AppServers.

The following example shows how to set up a NameServer instance. A port must be allocated to each NameServer instance:

```
[NameServer.NS1]
    srvrLogFile=<Log DIR>/NS1.ns.log
    environment=NS1
    autoStart=1
    portNumber=4091
    host=<qxtendhost>
```

Set up the properties in the `ubroker.properties` file for the NameServer that will control the QXI AppServers.

Verify NameServer settings by running:

```
$DLC/bin/nsconfig -f <property file> -i <NameServer>
```

Validate NameServer settings by running:

```
$DLC/bin/nsconfig -f <property file> -i <NameServer> -v
```

Parameter File Setup

The AppServer connects to the QAD Enterprise Applications databases based on values defined in a parameter file. This parameter file is similar to those used when starting a normal QAD Enterprise Applications session with some minor differences.

The parameter file can contain any of the client startup parameters that are required. It should not contain a `-p` (startup procedure) because this is controlled by the AppServer settings. The following is an example of a parameter file that is connecting to remote database servers:

```
-db mfgprod -ld qaddb -H hp40 -S mfg-server -N TCP -trig triggers
-db admprod -ld qadadm -H hp40 -S adm-server -N TCP
-db helpprod -ld qadhelp -H hp40 -S help-server -N TCP
-d mdy -yy 1920 -Bt 350 -c 30 -D 100 -mmax 3000 -nb 200 -s 63
```

The following is an example of a parameter file that is connecting to local database servers:

```
-db /mfgpro/db/mfgprod -ld qaddb -znotrim -trig triggers
-db  /mfgpro/db/hlpprod -ld qadhelp
-db  /mfgpro/db/admprod -ld qadadm
-d mdy -yy 1920 -Bt 350 -c 30 -D 100 -mmax 3000 -nb 200 -s 63
```

Progress AppServer Setup

The AppServers are also configured in the `ubroker.properties` file. By default, QXtend installer creates a Progress AppServer for SI API with the default name qasi_AS. The following details are just for reference in case you need to manually create the AppServer.

When creating a new AppServer outside of the Progress Explorer, a uuid (universal unique identifier) must be specified. The ID must be unique because the controlling NameServer uses it to identify the broker. To generate a uuid, use the following command:

```
$DLC/bin/genuuid
```

Copy the generated uuid into the `ubroker.properties` file.

The following sample shows how an AppServer definition in the `ubroker.properties` file would be configured for QXI. Bold type indicates settings that may require changes for the QXI AppServer setup.

```
#
# Sample properties for AppServer hosting QAD Enterprise Applications APIs.
# Most of these entries are shown for illustrative purposes only.
#
[UBroker.AS.siapi]
appserviceNameList=siapi
autoStart=1
brkrLoggingLevel=4
brokerLogFile=/qad/4gl/logs/siapi.broker.log
controllingNameServer=NS1
debuggerEnabled=1
operatingMode=Stateless
portNumber=7777
PROPATH=<QXIAdapter>,<QADApps>/lib/qra.pl,<QADApps>
srvrLogEntryTypes=ASPlumbing,DB.Connects
srvrLogFile=/qad/4gl/logs/siapi.server.log
srvrLoggingLevel=4
srvrStartupProc=mfaistrt.p
uuid=268ba6b18f508f07:590042c1:1127391c8de:-8000
```

Note The PROPATH setting for the Financials module service interface depends on where this instance of the Financials module is installed. The PROPATH must specify the location of the QXI adapter directory at the start. The PROPATH must also contain an entry for `qra.pl`, the service interface library.

The `controllingNameServer` entry should point to the NameServer that was defined in the previous section. See “Progress NameServer Setup” on page 242.

For detailed information on setting up Progress AppServer, see *Installation Guide: QAD QXtend*.

Modifying the PROPATH

If the AppServer PROPATH is changed and the AdminServer is running, stopping and starting the AppServer is not sufficient to pick up the new PROPATH. You must restart the AppServer, NameServer, and AdminServer.

Important QXtend AppServers set up for JIT Sequencing must have *JITSInstallDir*`/jpssrc/apiint/app` at the beginning of the PROPATH.

Verifying the Implementation

Verify the settings for the AppServer by running the following command and checking the entries:

```
$DLC/bin/asconfig -f <property file> -i <NameServer>
```

Validate the AppServer settings by running:

```
$DLC/bin/asconfig -f <property file> -i <NameServer> -v
```

Commands to Start Servers

Start the AdminServer on default ports:

```
proadsv -start
```

Start the AdminServer and specify the ports to use:

```
proadsv -start -port <AdminServer Port> -adminport <AdminServer to Database Port>
```

Start the NameServer:

```
nsman -port <AdminServer Port> -i <NameServer Name> -x
```

Start the AppServer:

```
asbman -port <AdminServer Port> -i <AppServer Name> -x
```

Commands to Query Servers

Query the NameServer:

```
nsman -port <AdminServer Port> -i <NameServer Name> -q
```

Query the AppServer:

```
asbman -port <AdminServer Port> -i <AppServer Name> -q
```

Commands to Stop Servers

Stop the AdminServer:

```
proadsv -stop
```

Stop the NameServer:

```
nsman -port <AdminServer Port> -i <NameServer Name> -e
```

Stop the AppServer:

```
asbman -port <AdminServer Port> -i <AppServer Name> -e
```

Chapter 21

QXtend Inbound with QAD Q/LinQ

This section provides information on using Q/LinQ with QXtend Inbound (QXI).

Overview

Q/LinQ is a tool set for building integrations with third-party or in-house applications for complex data exchange. It also provides infrastructure for administering and managing the data exchange between QAD Enterprise Applications and these external applications. See *External Interface Guide: QAD Q/LinQ* for details about installing and using Q/LinQ.

Q/LinQ and QXI provide overlapping features in terms of queuing messages to be processed as QAD Enterprise Applications transactions. However, the processing of data into QAD Enterprise Applications through QDocs is a more robust method than the more CIM-like user-interface-emulation approach generally used by earlier releases of Q/LinQ. If you have installed and configured Q/LinQ and now install QXI, you can use the services of QXtend to process messages received and managed by Q/LinQ through its processing API.

There are a number of reasons why you might want to use Q/LinQ and QXI together:

1. Q/LinQ does outbound processing, while QXtend is currently only available for inbound documents.
2. Q/LinQ has APIs to perform data mapping.
3. Q/LinQ has middleware communications capability and the associated feature of generating and processing confirmations.
4. Q/LinQ stores in-process documents in a database rather than externally as files. This approach provides enhanced data security and scalability and reduces the overall number of pieces that must be managed in the integration.

In addition, you can continue to process some Q/LinQ import documents using UI emulation, while redirecting processing responsibility to QXtend for others.

To process QDocs, you need to follow some special steps when you set up and configure Q/LinQ:

1. Install Q/LinQ based on standard installation instructions, giving special consideration to the setting in the `QqMomAdapter.ini` file that affects processing of QDocs.
2. Specify the location of the QXI server in Q/LinQ Control (36.8.24).
3. Define appropriate defaults in Register External Application (36.8.1.1).
4. Define import specifications for QDocs using Import Specification Maintenance (36.8.1.3).
5. Define import and export specifications for the acknowledgement documents passed between QAD Enterprise Applications and QXI.

Install and Configure Q/LinQ

You should install and configure Q/LinQ using the instructions in *External Interface Guide: QAD Q/LinQ* in conjunction with the information related to the most current service pack for Q/LinQ. To use Q/LinQ with QXtend, you must have one of the following minimum release levels:

- eB Service Pack 7 or higher
- eB2 Service Pack 4 or higher
- eB2.1 or QAD Enterprise Applications 2008 Standard Edition, any version

- QAD Enterprise Applications 2008 Enterprise Edition

A section in the Q/LinQ MOM adapter initialization file (`QqMomAdapter.ini`) supports the use of Q/LinQ with QXI. The `[envelope usage]` section indicates for each supported Q/LinQ external application ID whether it uses a QDoc envelope for inbound and outbound communications. Setting this parameter to True indicates to Q/LinQ that it should remove the SOAP envelope before processing a document received from a specific application ID.

Table 21.1 lists the variable in the `[envelope usage]` section of the Q/LinQ MOM Adapter.

Table 21.1
Envelope Section

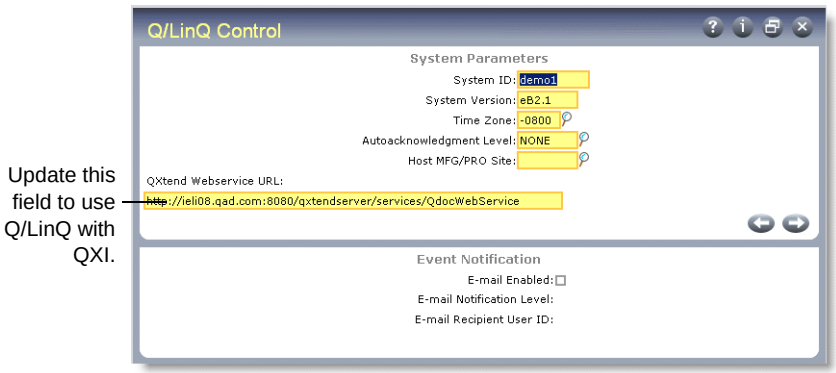
Variable	Description
<application ID value>	True if all messages received from and sent to the external application will use the QDoc envelope; otherwise, false. If omitted, false is assumed.

Setting the value to false or omitting it causes the Q/LinQ MOM adapter to behave as it did before the introduction of QDocs.

Define QXI URL

Specify the URL that identifies the location of QXtend in your system (maximum 70 characters) in Q/LinQ Control (36.8.24).

Fig. 21.1
Q/LinQ Control (36.8.24), QXtend Webservice URL



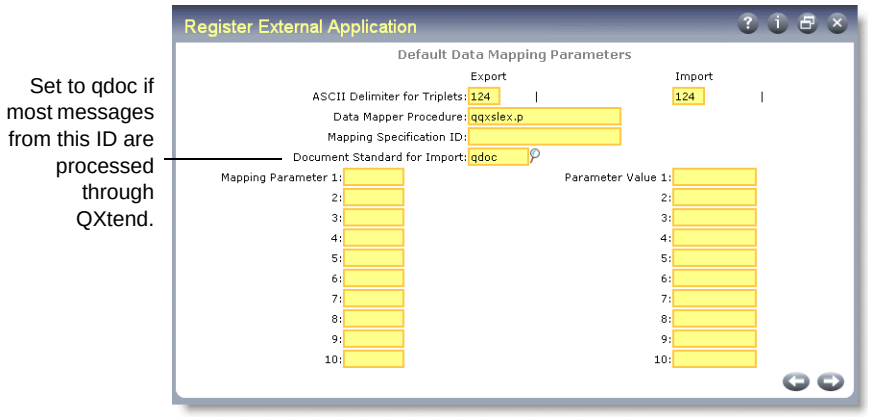
Q/LinQ uses this URL to locate the QXtend server when the import specification associated with a document indicates that it should be processed through QXtend.

Define External Application Defaults

In Register External Application (36.8.1.1), enter the unique IDs you have decided upon for identifying specific connections. Settings in two frames affect the relationship between Q/LinQ and QXtend.

In the Default Data Mapping Parameters frame, review the setting for Document Standard for Import.

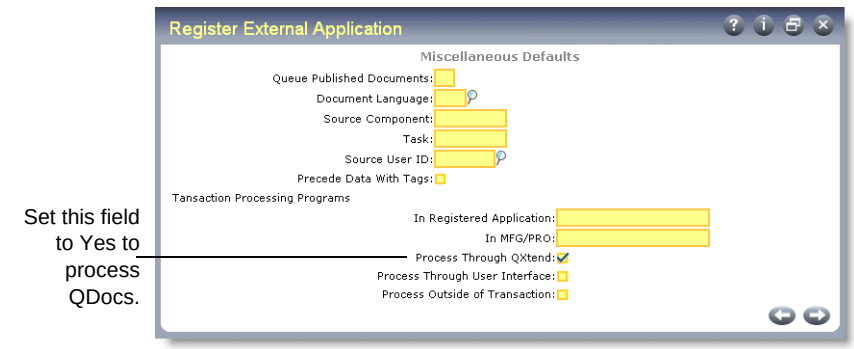
Fig. 21.2
Register External Application (36.8.1.1), Default Data Mapping Parameters



If most of the documents you are going to process using this connection will be mapped to QDocs, specify qdoc in this field. Q/LinQ uses this field to determine how to process a message when a specific import specification cannot be found.

In the Miscellaneous Defaults frame, set the appropriate value for Process Through QXtend.

Fig. 21.3
Register External Application, Miscellaneous Defaults



The value you set here defaults to the same field in Import Specification Maintenance when you define a new import specification for this application. If you plan to process most of the documents from this external application as QDocs, set this field to Yes.

Set Up Import Specifications for QDocs

Use Import Specification Maintenance (36.8.1.3) to set document-specific parameter values for importing messages to be processed as QDocs.

Many fields in this program default from the values you specify in Register External Application.

Matching Specifications to Documents

You can use up to five values to define an import specification: document standard, document type, document revision, application ID, and trading partner ID. The only required field is the document type. The system uses the following logic to find a specification to apply to a document:

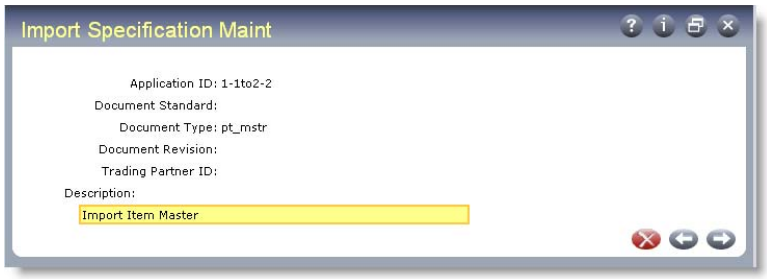
1. It looks for one with an exact match for document standard, document type, document revision, application ID, and trading partner ID.
2. It looks for one with matching document standard, document type, document revision, application ID, and a blank trading partner ID.
3. It looks for one with matching document standard, document type, document revision, and blank application and trading partner IDs.

This lets you set up generic specifications that can apply to all documents of a certain type (and optional standard and revision) regardless of the particular application or trading partner associated with a document.

Register Import Specifications

The key field in Import Specification Maintenance (36.8.1.3) related to QXtend is Process Through QXtend.

Fig. 21.4
Import Specification Maintenance (36.8.1.3)



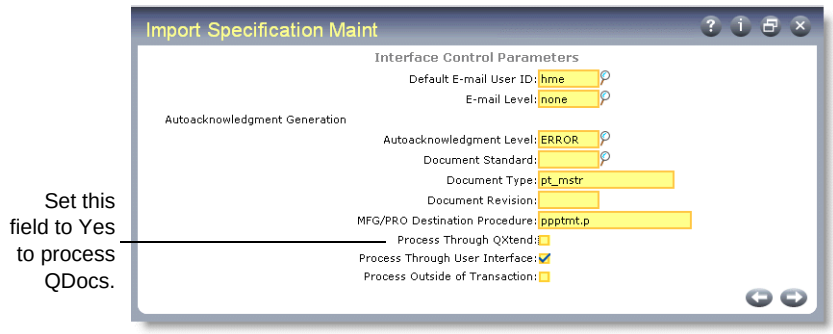
Application ID. Enter the name of an external application. Leave blank if you want this specification to apply to all documents of a certain type, standard, and revision without regard to the associated application.

Document Standard. Enter qdoc as the document standard to process documents through QXtend.

Document Type. Enter a user-defined name for the type of data that is being processed.

Document Revision and Trading Partner ID can be left blank. If they are used, Document Standard, Document Type, Document Revision, and Trading Partner ID must be a unique combination of values.

Fig. 21.5
Import Specification Maintenance, Interface Control Parameters



In the Interface Control Parameters frame, set up incoming processing parameters.

MFG/PRO Destination Procedure. Leave this field blank to process the incoming document through QXtend. When Process Through QXtend is Yes, the system automatically invokes `qqqdocpr.p` to process the imported document as a QDoc.

Process Through QXtend. Indicate if you want the documents associated with this import specification to be processed using QXI.

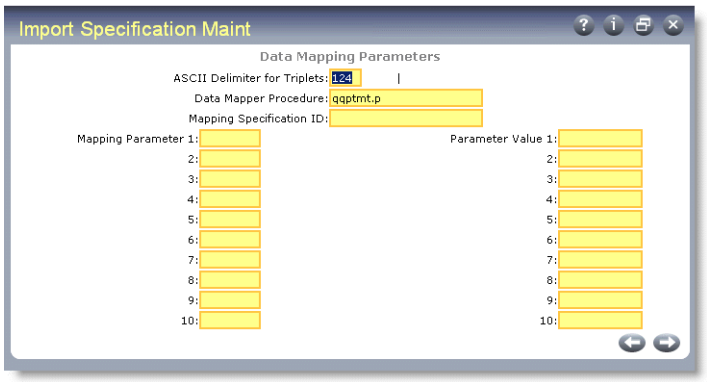
No: Documents are processed through user interface emulation or gateway programs.

Yes: Documents are processed as QDocs through QXI.

Note When this field is Yes, the document is treated as a QDoc regardless of the setting for Process Through User Interface.

Process Through User Interface. Enter Yes to invoke the destination procedure through the CIM Interface. When Process Through QXtend is Yes, this field is assumed to be No.

Fig. 21.6
Import Specification Maintenance, Data Mapping Parameters



In the Data Mapping Parameters frame, enter the user-written mapping program that maps the input data to the QDoc XML format.

Set Up for Acknowledgements

When you use Q/LinQ and QXtend together, QDoc confirmation documents can be sent between the two applications to acknowledge the receipt and processing of messages. Specific mapping procedures and XSLT stylesheets must be used to map these documents to the format required by the receiving application.

Use the values in Table 21.2 in the Data Mapping Parameters frame of Export Specification Maintenance (36.8.1.2) to map QDoc confirmations to the CONFIRM_BOD format used by Q/LinQ.

Table 21.2
Mapping Values for Exporting Acknowledgements

BOD:	`CONFIRM_BOD`
XSLT Stylesheet (Mapping Specification ID):	`QdocToCoBo4e.xsl`
DTD File:	`002_confirm_bod_004.dtd`
Data Mapper Procedure:	`qqxslex.p`
Mapping Parameter 1:	`rawform`
Parameter Value 1:	`xml`

Use the values in Table 21.3 when setting up records in the Interface Control Parameters and Data Mapping Parameters frames of Import Specification Maintenance (36.8.1.3) to map Q/LinQ CONFIRM_BOD messages to the XML QDoc format.

Table 21.3
Mapping Values for Importing Acknowledgements

BOD:	`CONFIRM_BOD`
XSLT Stylesheet (Mapping Specification ID):	`CoBo4iToQdoc.xsl`
DTD File:	`002_confirm_bod_004.dtd`
Data Mapper Procedure:	`qqxslin.p`
MFG/PRO Destination:	`qqprccnf.p`
Process Through User Interface:	`No`
Mapping Parameter 1:	`rawform`
Parameter Value 1:	`xml`

Section 3

QAD QXtend Licensing

This section contains information on implementing and using licensing in QXtend.

Chapter 22

Licensing

Introduction

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

QAD QXtend licensing is based on a hierarchy of the base module, receivers, and agents. Each is individually priced. The base module requires a number of receivers and agents, and each agent is coupled with an additional receiver. The base module becomes available when you enter a valid license. The license encodes how many receivers and agents are available for each QAD QXtend installation, the type of license, and a unique code indicating the organization who owns the license.

Agents move data into and out of the system. They listen for requests to transmit data, gather the data to transmit, and deliver it to the receiver. In QXI, connection pools are assigned one or more processing threads. When the system processes a QDoc request, it uses an available connection pool agent. In QXO, 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 QAD QXtend installation. The number of receivers is not enforced.

Licensing Types

There are four types of licensing:

- QAD & Approved QAD Partner Free Use
- Standard
- Enterprise
- Message-based

QAD & Approved QAD Partner Free Use

Under this default license type, no license code has been loaded. Any messages between QXO and QXI are carried free of charge regardless of whether the messages are data synchronization or not. In addition, messages between other QAD applications are carried free of charge using QAD QXtend QAD & Approved QAD Partner Free Use license. For example, QAD Production Scheduler uses QAD QXtend to update work orders. Partner applications also use QAD QXtend messaging free of charge.

For details regarding the free use of QXO, see “Free Use of QXO” on page 39.

Standard

Standard licensing limits the number of agents and receivers that can be used across QAD 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.

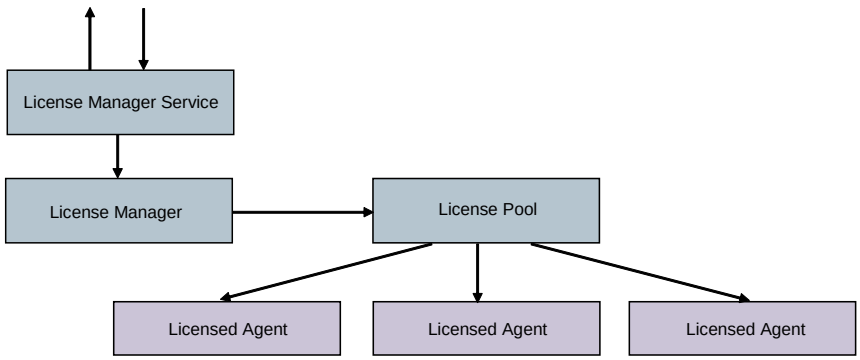
License Manager

QAD QXtend pricing and licensing is enforced using the License Manager Web service, which controls the release of receivers and agents. The License Manager enables you to limit the total number of licensed agents, monitor the release of licenses, and track the number of receivers in use. It is located within QXI and supports a single instance of QAD QXtend.

The License Manager enforces the agent limit encoded in the QAD 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 QAD QXtend installation.

When the License Manager is started, it creates a license pool for the current QAD 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.

Fig. 22.1
License Manager Flow



Each time a Message Sender service in QXO is started for an unregistered subscriber, the system makes a call to the License Manager to reserve an agent for the task. In QXI, 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 QXI and QXO services.

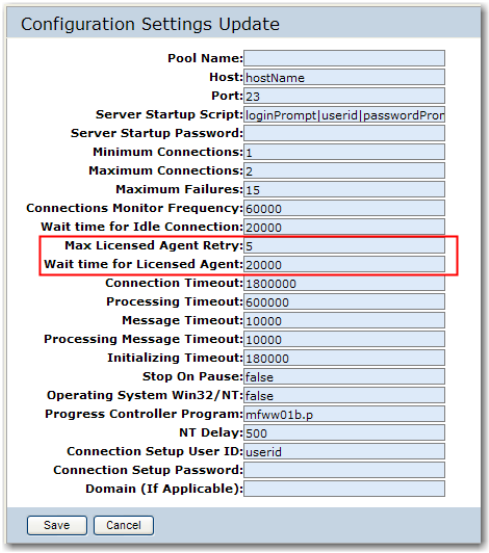
A connection thread is an idle connection in the QXI connection pools or a Message Sender thread in QXO. Each time a connection thread is closed, the License Manager releases the agent from the reserved pool.

License 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.

Fig. 22.2
Connection Pool Settings



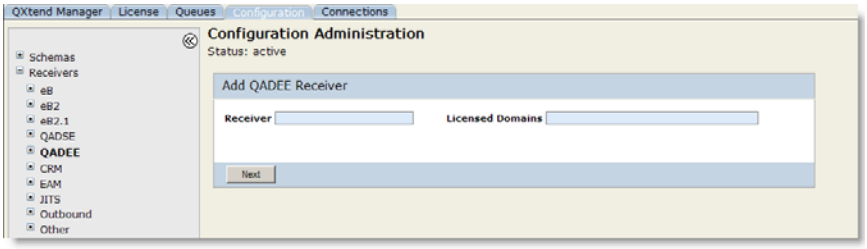
Max Licensed Agent Retry. Specify the number of times the system attempts to reserve a licensed agent before returning an exception.

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.

Fig. 22.3
Add Receiver, Licensed Domains



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.

Applications without Domains

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.

Fig. 22.4
Add eB Receiver

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 QAD QXtend.

Each time you configure a subscriber (receiver) in QXO, the system sends a message to the License Manager.

Fig. 22.5
License Manager Details

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.

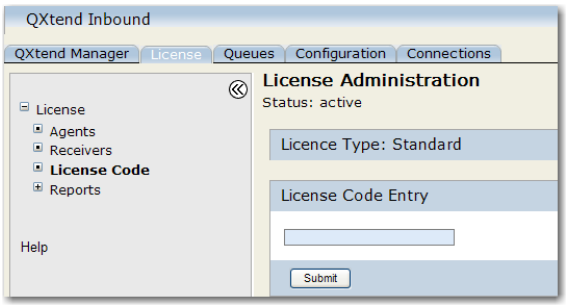
Number of Registered Receivers. Displays the number of registered receivers in both Inbound and Outbound.

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.

Fig. 22.6
License Administration



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 updates the QXtend license type accordingly. If a license code has already been entered, the system overwrites it based on the new license code.

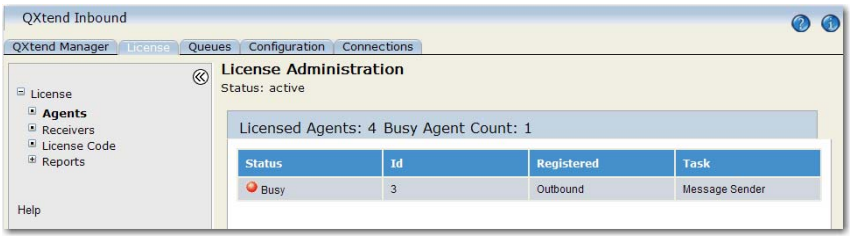
If a license file has already been entered, the system overwrites it and resets the License Manager's internal count information.

Agent and Receiver Statuses

Licensed Agents

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.

Fig. 22.7
Licensed Agents



Status. Displays the status of the agent. For example, Busy or Idle.

ID. Displays the process ID of the agent.

Registered. Indicates where the agent is registered, QXI or QXO.

Task. If processing a message through QXI, the name of the QDoc currently being processed is displayed. If the agent is assigned to QXO, the task is the Message Sender, which indicates that the agent is currently occupied by a Message Sender service in QXO.

Licensed Receivers

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

Fig. 22.8
Licensed Receivers

Name. Displays the name of the receiver.

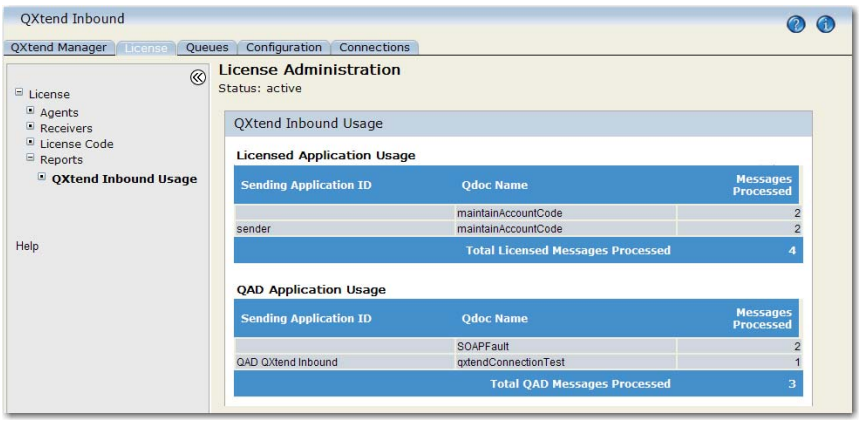
Registered. Indicates where the agent is registered, QXI or QXO.

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.

Fig. 22.9
QXtend Inbound Usage Report



Section 4

Reference

This section provides reference information.

Chapter 23

QDoc Structure Reference

This section provides reference information about the structure of various files associated with a QDoc.

Introduction

QDocs are inbound, XML, SOAP-compliant, data documents for target QAD Enterprise Applications instances. A QDoc is not accepted by QXI, however, unless it validates at least one schema document—for QDocs using the 1.1 syntax—or two schema documents if the QDoc uses the 1.0 syntax. In addition, if using the UI adapter, a QDoc is not entered into QAD Enterprise Applications unless an events file exists for the target program. Events files are not used for service interface APIs or the JITS API adapter.

QDoc schemas and events files are generated in the QGen utility from a QGen-built data file.

The following sections show examples of each of the various QDoc components—data file, QDoc schema, and target program events file.

QDoc Data Files

The QDoc data file is generated in QGen. The following is an excerpt of the `maintainSalesOrder.dat` file using the QDoc 1.0 specification. The bold numbers are QGen sequence numbers and are the beginnings of lines. Lines where three periods appear (...) show where lines have been removed for simplicity. A description of relevant document contents follows the file excerpt. See Chapter 18, “QGen,” on page 215 for details.

```
1000,"salesOrder",1,"Start/End Iteration","f4","a","so_nbr",
"soNbr#0:a021","soNbr","Order",yes,no,0,"x(8)","CHARACTER",
"so_mstr","RETURN",yes,no,no,no,"","","",yes,yes, "iedPortTransh#0:a051"
2000,"",1,"None","","a","so_cust","soCust#0:a021","soCust", "Sold-
To",no,no,0,"x(8)","CHARACTER","so_mstr","RETURN",yes,no, no,no,"","","",yes,yes,""
3000,"",1,"None","","a","so_bill","soBill#0:a021","soBill", "Bill
To",no,no,0,"x(8)","CHARACTER","so_mstr","RETURN",yes,no, no,no,"","","",yes,yes,""
4000,"",1,"None","","a","so_ship","soShip#0:a021","soShip", "Ship-
To",no,no,0,"x(8)","CHARACTER","so_mstr","RETURN",yes,no, no,no,"","","",yes,yes,""
4100,"",1,"None","",?,"yn","yn#0:1","yn",?,no,no,0,"yes/no",
"LOGICAL",?,"RETURN",yes,no,no,no,"","","",yes,yes,""
4200,"",1,"None","","new_ship","ad_name",
"adName#0:new_ship041","adName","Name",no,no,0,"x(28)",
"CHARACTER","ad_mstr","RETURN",yes,no,no,no,"","","",yes,yes,""
4300,"",1,"None","","new_ship","ad_line1",
"adLine1#0:new_ship041","adLine1","Address",no,no,0,"x(28)",
"CHARACTER","ad_mstr","RETURN",yes,no,no,no,"","","",yes,yes,""
...

58000,"transComment",2,"Start/End Iteration","f4","cmmt01", "cd_ref",
"cdRef#0:cmmt01021","cdRef","Master Reference",no,no,
0,"x(40)","CHARACTER","cd_det","return",yes,no,no,no,"","","",
yes,yes,"cdRef#0:cmmt01021"
59000,"salesOrderDetail",2,"Start/End Iteration","f4:f4","c",
"line","line#0:c051","line"," Ln",yes,no,0,">>9","INTEGER",?,
"return",yes,no,yes,no,"f4","S","return",yes,yes,
"cdType#0:cmmt01021"
59100,"allocationDetail",3,"Start/End Iteration","f4","alloc",
"lad_loc","ladLoc#0:alloc1091","ladLoc","Location",yes,no,0,
"x(8)","CHARACTER","lad_det","RETURN",yes,no,no,no,"","","",
yes,yes,"ladQtyAll#0:alloc1091"
59200,"",3,"None","","alloc","lad_lot","ladLot#0:alloc1091",
"ladLot","Lot/Serial",no,no,0,"x(18)","CHARACTER","lad_det",
"RETURN",yes,no,no,no,"","","",yes,yes,""
...
68400,"features",3,"Start/End Iteration","f4","w",
"work2_feature","work2Feature#0:w491","work2Feature","Feature",yes,no,0,"-
>>>,>>9.9<<<<<<","CHARACTER","work2_list","RETURN"
,yes,no,no,yes,"","","",yes,yes,"work2NetPrice#0:w091"
68500,"configuredItemDetail",4,"Start/End Iteration","f4",
"w","work2_comp","work2Comp#0:w091","work2Comp","Item Number" ,yes,no,0,"-
```

```
>>>,>>9.9<<<<<<","CHARACTER","work2_list","RETURN"
,yes,no,no,yes,"","","",yes,yes,"work2NetPrice#0:w091"
...
145000,"",2,"None","","a","ied_region","iedRegion#0:a051", "iedRegion","Region of
Origin/Dest",no,no,0,"x(3)","CHARACTER"
,"ied_det","RETURN",yes,no,no,no,"","","",yes,yes,""
146000,"",2,"None","","a","ied_port_transh","iedPortTransh#0:a051","iedPortTransh","Po
rt of Transshipment",no,no,0,"x(8)",
"CHARACTER","ied_det","RETURN",yes,no,no,no,"","","",yes,yes,""
```

Each sequence number represents a field. The sequence numbers reflect the program hierarchy. The numbers ascend in increments of 1000 if the fields are in their own frames or require the Go key to proceed. At 4000, where a separate frame is encountered, the fields within the frame ascend in increments of 100. A frame encountered inside this frame would increment in 10’s, and another inside that would increment in 1’s.

The first entry (1000) includes a name (salesOrder), the iteration (1), and the identifier (Start/End Iteration). It also includes the key required to leave the iteration (F4).

Other entries do not include names until 58000. This is the standard Transaction Comments entry in QDoc specification 1.0. It starts and ends an iteration, and so is iteration number 2. The next entry, 5900, named salesOrderDetail, is also iteration 2. This is because the comments iteration is already complete by the time you reach this field. The allocationDetail iteration, 59100, and all the fields associated with it, is numbered iteration 3 because it occurs inside the salesOrderDetail iteration.

The other iterations in the program—features and configuredItemDetail— are shown. Then there is a break until the last two entries in the file.

QDoc Schemas

QDoc schemas are a product of the QGen generate process.

Note The QDoc 1.1 schema differs from the 1.0 schema. This section contains information about both schemas. Refer to the appropriate section.

See Chapter 24, “QDoc Specifications and Standards,” on page 275 for details about how to name QDocs and the elements included in the schema files.

QDoc Schema 1.1

For QDoc syntax specification 1.1 one schema is created. For the first version of maintainSalesOrder QDoc on QAD EE, this would be titled `maintainSalesOrder-eB2_2.xsd`.

QGen combines the name of the QDoc with the target QAD Enterprise Applications version to create the file names.

Flat Structure

Schemas defined in the 1.0 Qdoc standard use multiple levels of include and import elements to allow a level of abstraction and make it easier to read and analyze the schemas. Because Progress OpenEdge has problems importing some XML schemas in this format, Qdoc schemas in the 1.1 standard use a flat structure resulting in schemas being self-contained.

Complex Types

In the 1.1 specification, the `complexContent` and `extension` elements have been removed. The `annotation` and `documentation` elements also have been removed due to a restriction in the number of annotations that can be read from a schema.

```
<complexType name={name}>
    <sequence>
        <element name={element name} type={type}/>
    <sequence>
</complexType>
```

Keys and Relationships

In the 1.0 QDoc standard relationships between tables cannot be defined, only inferred. Primary keys for each iteration are stored in the `complexType` element.

To create datasets from XML schema, primary keys and relationships must be defined in the schema. Standard XML schema notation is defined as:

```
<unique name="PK_{complex type name}"/>
    <selector xpath=".//{complex type name}"/>
    <field xpath="{primary key field name}"/>
</unique>
```

Multiple primary key fields are defined by adding more field elements. Relationships between tables are defined as:

```
   <keyref name="{complex type name}" refer="qdoc:{unique name of parent}" prodata:nested=
"true">
    <selector xpath=".//{child complex type name}"/>
        <field xpath="{common field name}"/>
   </keyref>
```

The name attribute of `keyref` is only used by .NET when building datasets—OpenEdge does not require it.

To support relationships such as these, the primary key of the parent table must also be included in the child table. This is not the case with the 1.0 Qdoc standard.

Transaction Comments

The XML schema for the transaction comments resides in `qdocCommon-eB_1.xsd`. With the 1.1 schemas having a flat structure, this file is no longer required to be accessible.

In 1.0 the iteration names for the transaction comments always started with `transComment`. To be able to create a dataset from a schema, for each iteration—equivalent to a table—the name must be unique. Therefore all node names of `transComment` now have a prefix of the parent table name. For example, transaction comments for the `<salesOrder>` node is now `<salesOrderTransComment>`.

Session Context

In the QDoc 1.1 syntax specification, session context information is not included in the schema of each QDoc. This is due to a limitation when creating dataset schema from an XML schema that it cannot have a nested dataset.

Consequently the session context schema is excluded from the QDoc schema but included in the WSDL. Automated tools to generate datasets from XML schema to create QDocs using standard Progress features must take into account that the session context must be added separately.

`dsSessionContext` has the following structure:

```
<dsSessionContext>
    <ttContext>
        <propertyQualifier/>
        <propertyName/>
        <propertyValue/>
    </ttContext>
</dsSessionContext>
```

Application-specific parameters are now passed as part of the `dsSessionContext` structure. This structure includes the following attributes:

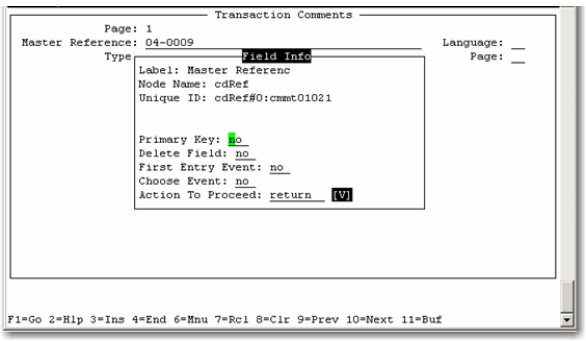
- `domain`. Domain has been moved to `dsSessionContext` as a property name of `domain`. Previously it was a parameter in the URL in the `receiverId`.
- `scopeTransaction`. This attribute has been added to `dsSessionContext` as a property name of `scopeTransaction`. For details see “scopeTransaction” on page 287.
- `documentVersion`. This attribute has been moved to `dsSessionContext` as a property name of `version`. Previously it was part of the root node. For details see “documentVersion” on page 305.
- `mnemonicsRaw`. This attribute has been added to `dsSessionContext` as a property name of `mnemonicsRaw`. For details see “mnemonicsRaw” on page 286.
- Requestor `id`. This attribute of `requestor` has been moved to `dsSessionContext` as a property name of `username`. For details see “Requestor Element” on page 302.
- Requestor `password`. This attribute of `requestor` has been moved to `dsSessionContext` as a property name of `password`. For details see “Requestor Element” on page 302.
- Arrays. For details see “Array Representation” on page 283.

Note For Service Interface APIs, the session context is returned to the caller. This helps to speed up processing, since the session context will contain the Session ID code, and this can be extracted from the response and re-used in subsequent requests. Username/password authentication is slightly slower than when providing a valid Session ID code in the request.

Mapping Transactions Comments

When you enter the Transaction Comments section, use the following process to map the screen:

1 Press Enter in the Master Reference field.



QDoc Schemas 1.0

In QDoc syntax specification 1.0 two schemas are created, a base schema and a type schema. For the first version of maintainSalesOrder QDoc on eB, these would be titled:

- `maintainSalesOrder-eB_1.xsd`
- `salesOrderType-eB_1.xsd`

See Chapter 24, “QDoc Specifications and Standards,” on page 275 for details about how to name QDocs and the elements included in the schema files.

Base QDoc Schema

The base QDoc schema primarily identifies the name and location of the type QDoc schema. An incoming QDoc is validated against this base schema for agreement with the existing QDoc schemas. The following is the entire base QDoc schema, `maintainSalesOrder-eB_1.xsd`.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <schema xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:enc="http://www.w3.org/2002/12/soap-encoding"
    xmlns:qdoc="http://www.qad.com/qdoc/eb"
    xmlns:qcom="http://www.qad.com/qdoc/common"
    targetNamespace="http://www.qad.com/qdoc/eb"
    xml:lang="EN">
      <include schemaLocation="salesOrderType-eB_1.xsd" />
    - <element name="maintainSalesOrder">
        - <complexType>
        - <sequence>
          <element name="salesOrder" type="qdoc:SalesOrderType"
           minOccurs="1" maxOccurs="unbounded" />
          </sequence>
          <attributeGroup ref="qcom:commonAttributes" />
          </complexType>
      </element>
  </schema>
```

Type QDoc Schema

The type QDoc schema contains the program hierarchy and the field and data requirement details. Lines where three periods appear (...) show where lines have been removed for simplicity. A description of relevant document contents follows the file excerpt.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <schema targetNamespace="http://www.qad.com/qdoc/eb"
    xmlns:qcom="http://www.qad.com/qdoc/common"
    xmlns:qdoc="http://www.qad.com/qdoc/eb"
    xmlns:enc="http://www.w3.org/2002/12/soap-encoding"
    xmlns="http://www.w3.org/2001/XMLSchema" xml:lang="EN">
      <include schemaLocation="qdocCommon-eB_1.xsd" />
    - <complexType name="SalesOrderType" qdoc:primaryKeys="soNbr">
    - <complexContent>
    - <extension base="qdoc:ApiTempTableType">
        - <sequence>
          <element name="operation" type="qdoc:OperationType" minOccurs="0" />
        - <element name="soNbr" type="string" minOccurs="0">
        - <annotation>
          <documentation>Order</documentation>
          </annotation>
          </element>
        - <element name="soCust" type="string" minOccurs="0">
        - <annotation>
          <documentation>Sold-To</documentation>
          </annotation>
        ...
        </element>
        - <element name="ladQtyAll" type="decimal" minOccurs="0">
        - <annotation>
          <documentation>Qty Alloc</documentation>
          </annotation>
          </element>
          </sequence>
      </extension>
      </complexContent>
      </complexType>
  </schema>
```

Each element is a field in the target QAD Enterprise Applications program. No validation is required in the XML file because the validation in QAD Enterprise Applications remains active during a QDoc entry session.

Events Files

Events files contain navigation requirements for a target QAD Enterprise Applications program and by default are named the same as the QAD Enterprise Applications procedure with a .xml extension. The sales order entry program in QAD Enterprise Applications is `sosomt.p`; the events file for Sales Order Maintenance is `sosomt-eB2_1.xml`.

Events File Example (1.1)

The following is a partial `sosomt-eB2_1.xml` file in the QDoc 1.1 specification. Lines where three periods appear (...) show where lines have been removed for simplicity. A description of relevant document contents follows the file.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <apicontroller xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=
"http://ieli08.qad.com/apimodel/xmlenginexml/events/apicontroller.xsd">
    <program name="String" mfgprobase="String" delta-version="String" description=
    "String" default-operation="M" />
    - <iterations>
        - <field uid="soNbr#0:a021" name="so_nbr" inheritevents="true">
        <IterationEvent iterationname="salesOrder" exititeration="f4" />
        </field>
        - <field uid="cdRef#0:cmmt01021" name="cd_ref" inheritevents="true">
          <IterationEvent iterationname="salesOrderTransComment" exititeration="f4" />
          </field>
        ...
        - <field uid="iedLine#0:a051" name="ied_line" inheritevents="true">
        <IterationEvent iterationname="orderLineDetail" exititeration="f4" />
        </field>
        </iterations>
        - <operation action="R">
        - <events>
        - <field uid="soOrdDate#0:b0121" name="so_ord_date" inheritevents="false">
        <DeleteEvent promptfield="delYn#0:1" sendvalue="true" actionkey="return" />
        </field>
        ...
        - <field uid="sodlcRef#0:line_det0101" name="sodlc_ref" inheritevents="false">
        <DeleteEvent promptfield="delYn#0:1" sendvalue="true" actionkey="return" />
        </field>
        </events>
        </operation>
        - <operation action="A,M,S">
        - <events>
        - <field uid="line#0:c051" name="line" inheritevents="true">
        <FirstEntryEvent prekey="f4" sendvalue="true" postkey
        "return" />
        </field>
        - <field uid="work2Feature#0:w491" name="work2_feature" inheritevents="false">
        <ChooseEvent />
        </field>
        - <field uid="work2Comp#0:w091" name="work2_comp" inheritevents="false">
        <ChooseEvent />
        </field>
        </events>
        </operation>
    - <mappings>
        <field-map uid="soTaxc#0:set_tax1881" nodename="soTaxc1" />
        <field-map uid="soTaxable#0:set_tax1881" nodename="soTaxable1" />
        <field-map uid="pMsgConfirm#0:2" nodename="pMsgConfirm1" />
```

```
        <field-map uid="pMsgConfirm#0_INTRANS:1" nodename="pMsgConfirm2" />
        <field-map uid="pMsgConfirm#0_INTRANS:2" nodename="pMsgConfirm3" />
        <field-map uid="sodSite#0:c_btb_site1551" nodename="sodSite1" />
        <field-map uid="sodTaxc#0:set_tax17131" nodename="sodTaxc1" />
        <field-map uid="sodTaxable#0:set_tax17131" nodename="sodTaxable1" />
        <field-map uid="pMsgConfirm#0_CONS2:2" nodename="pMsgConfirm1" />
        <field-map uid="pMsgConfirm#0_INTRANS2:1" nodename="pMsgConfirm2" />
        <field-map uid="pMsgConfirm#0_INTRANS2:2" nodename="pMsgConfirm3" />
    </mappings>
</apicontroller>
```

The first set of entries in the events file are the iterations. The first iteration starts on the field `so_nbr` for the iteration salesOrder. The second iteration is the transaction comments iteration in the same order as in the data file. In QDoc syntax specification 1.1 each transaction comment section—if a child of a different parent—has a different node name. Therefore the parent node is used as a prefix to `TransComment`; for example `salesOrderTransComment`. Each iteration is identified by QAD Enterprise Applications field, iteration name, and the key required to exit the iteration.

Following the iterations is the delete event and the first entry event fields. Delete fields require a field name, the value to send to confirm the deletion, and the key to complete the deletion.

The mappings section contains the mappings between the unique identifiers that are assigned by QGen and the node names. While the nodenames are editable, the field-map IDs are not.

Events File Example (1.0)

The following is an entire `sosomt.xml` file in the QDoc 1.0 specification. A description of relevant document contents follows the file.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <apicontroller xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://ieli08.qad.com/apimodel
  /xmlenginexml/events/apicontroller.xsd">
    <program name="String" mfgprobase="String" delta-version="String" description="String" default-
    operation="M" />
    - <iterations>
        - <field uid="soNbr#0:a02" name="so_nbr" inheritevents="true">
        <IterationEvent iterationname="salesOrder" exititeration="f4" />
        </field>
        - <field uid="cdRef#0:cmmt0102" name="cd_ref"
           inheritevents="true">
        <IterationEvent iterationname="transComment" exititeration="f4" />
        </field>
        - <field uid="line#0:c05" name="line" inheritevents="true">
        <IterationEvent iterationname="salesOrderLineDetail" exititeration="f4:f4" />
        </field>
        - <field uid="work2Comp#0:w09" name="work2_comp" inheritevents="true">
        <IterationEvent iterationname="configurationDetail" exititeration="f4:f4" />
        </field>
        - <field uid="ladLoc#0:alloc109" name="lad_loc" inheritevents="true">
        <IterationEvent iterationname="allocationDetail" exititeration="f4" />
        </field>
        - <field uid="cmtSeq#0:cmmt0102" name="cmt_seq" inheritevents="true">
        <IterationEvent iterationname="transComment" exititeration="f4" />
        </field>
        - <field uid="taxLine#0:b05" name="tax_line" inheritevents="true">
        <IterationEvent iterationname="taxDetail" exititeration="f4" />
        </field>
        - <field uid="iedLine#0:a05" name="ied_line" inheritevents="true">
        <IterationEvent iterationname="orderLineDetail" exititeration="f4" />
        </field>
        </iterations>
        - <operation action="R">
        - <events>
        - <field uid="soDueDate#0:b012" name="so_due_date" inheritevents="false">
        <DeleteEvent promptfield="delYn#0" sendvalue="y" actionkey="return" />
        </field>
        - <field uid="line#0:c05" name="line" inheritevents="true">
        <FirstEntryEvent prekey="f4" sendvalue="S" postkey="return" />
        </field>
        </events>
        </operation>
        - <operation action="A,M,S">
        - <events>
```

```
            - <field uid="line#0:c05" name="line" inheritevents="true">
            <FirstEntryEvent prekey="f4" sendvalue="S" postkey="return" />
            </field>
            </events>
        </operation>
    </apicontroller>
```

The first set of entries in the events file are the iterations. The first iteration starts on the field `so_nbr` for the iteration salesOrder. The second iteration is the transaction comments iteration in the same order as in the data file. Each iteration is identified by QAD Enterprise Applications field, iteration name, and the key required to exit the iteration.

Following the iterations is the delete event and the first entry event fields. Delete fields require a field name, the value to send to confirm the deletion, and the key to complete the deletion.

Chapter 24

QDoc Specifications and Standards

This section describes the structure and content of QDocs supported by QAD QXtend Inbound (QXI).

Important Users who want to build their own QDocs should follow these guidelines to conform as closely as possible to the design standards implemented by QAD.

QDoc Naming and Identification

The material presented in this section assumes a working knowledge of basic XML syntax, XML namespaces, and XML schema. While many of the points pertain specifically to QAD Enterprise Applications, QAD plans to apply most of the same rules to QDocs used by other QAD application products in future releases of QAD QXtend.

Note In order to support the new service interface layer—as well as the adoption by Progress of XML notation in OpenEdge 10.1—significant revisions have been made to the QDoc syntax specification, resulting in a 1.1 specification. Most components in QAD QXtend—with the exception of the service interface API which supports only the 1.1 syntax—support both the 1.0 and 1.1 versions of the QDoc syntax. The information contained in this section applies to both QDoc standards, except where indicated.

Common QDoc Naming Convention

Each QDoc name consists of an action (verb) followed by its object (noun) that best expresses the meaning of the QDoc. The name should be as similar as possible to the corresponding API method name, if any, that implements the business logic required for the QDoc.

Important The length of the name of a QDoc can be a maximum of 32 characters.

Table 24.1 describes QDoc naming conventions.

Table 24.1
QDoc Name Suffixes

QDoc Type	Suffix	Examples
Request In	-	maintainSalesOrder
Response Out	Response	maintainSalesOrderResponse

The suffix for the Response Out type is assigned according to present WSDL message-naming conventions, where requests are customarily not suffixed and responses are customarily suffixed with the literal Response.

All QDoc names are written in mixed-case or camel-case notation for Java classes, in which the first letter of each token within a name is capitalized, while all other letters are lower-case.

In the case of QAD Enterprise Applications, the `maintain` verb is used for all create-read-update-delete (CRUD) type QDocs with the detailed action (add, modify, delete, sync, get) expressed in the body of the QDoc.

Common Message Envelope

In order to route QDocs between applications, all QDocs should be wrapped inside a standard message envelope that describes the QDoc's source and destination, among other data of interest. Unlike the application-specific QDoc content with details that vary, the message envelope has a fixed, fully specified syntax that applies to all QDocs.

The envelope is designed to comply with the SOAP 1.0 and 1.1 standards, and to translate relatively easily into emerging industry messaging standards such as Java Message Service (JMS) and Electronic Business XML (ebXML). It can be used with non-QDoc documents and must be used with QDocs. See "QDoc Message Envelope" on page 301.

The design of the QDoc message envelope most likely will change in future releases, as SOAP envelope header standards are promulgated and adopted by the software industry. The QDoc envelope will evolve to comply with emerging Web-service standards.

QDoc Namespaces

All QDoc and QDoc components are defined in some XML namespace, in order to clearly identify the QAD application product with which they are associated as well as prevent QDoc naming clashes across QAD products. The namespace URIs are abstract identifiers that do not dictate the URL, directory, or path names in which the QDoc files must reside.

All QAD-standard QAD Enterprise Applications QDocs use one of the following namespace URIs, depending on the QDoc syntax specification:

- QDoc 1.0 specification: `http://www.qad.com/qdoc/eb`
- QDoc 1.1 specification: `urn:schemas-qad-com:xml-services`

QDoc components common to all QAD products use one of the namespace URIs listed below. In particular, the QDoc message envelope and QDoc confirmation are defined in this namespace:

- QDoc 1.0 specification: `http://www.qad.com/qdoc/common`
- QDoc 1.1 specification: `urn:schemas-qad-com:xml-services:common`

Users building custom QDocs or extending QAD-defined QDocs must define their new XML elements and attributes inside a different namespace than the QAD namespaces to allow the unambiguous mixing of QAD- and user-defined syntax within the same QDoc. Separate namespaces also allow custom QDoc syntax to be validated using separate, user-specific XML schemas.

The standard QDoc namespace URI incorporates the URL of the QAD Web site (`http://www.qad.com`). For custom QDocs, you can use the simpler URN syntax, which is colon-delimited beginning with the `urn:` prefix as in the following examples:

```
urn:schemas-microsoft-com:xml-msdata
urn:schemas-progress-com:xml-prodata:0001
urn:schemas-qad-com:xml-services
```

The tokens between each colon are user-defined. The last token can be used as a version identifier, as in the second example above.

In QDoc specification 1.0 various predefined elements and attributes used across all QDoc types are defined in the namespaces and placed into a single XML schema file called `qdocCommon-<version>.xsd`. See “version” on page 286. These common XML schema files do not contain application objects, but only non-application data. The appropriate version is imported into every QAD Enterprise Applications QDoc XML schema.

In QDoc specification 1.1 versioning information is contained within the schema file name. See “Schema File Names and Versioning” for details.

Schema File Names and Versioning

Versioning is handled differently depending on the syntax specification in use. Refer to the appropriate section.

Versioning for 1.1 Schemas

The XML schema files defining each QDoc, both requests and responses, are named `<QDoc name><Response>-<version>.xsd`, where `<version>` is the value of the QDoc version attribute. See “version” on page 286. The Response component is appended on response documents only.

Versioning for 1.0 Schemas

For all QAD products, it is expected that XML schemas will be required for each reusable data type, such as `SalesOrderType` or `PurchaseOrderLineType`, as well as for the QDocs themselves. Because the definitions of the data types will change over time, the XML schema files of the data types are versioned, with different versions of the same data type coexisting in different XML schema files. In fact, it is anticipated that most QDoc version changes will be driven by changes in the underlying data types.

The XML schema defining each version of a single QDoc, therefore, includes the XML schema files for the corresponding versions of its component data types. See “version” on page 286. To allow the data type schemas to be referenced by version, each XML schema file defining a data type is named `<data type>-<version>.xsd`, where `<version>` is assigned in exactly the same manner as for the QDocs.

As a general rule, whenever any common data type is revised and obtains a new version, all the QDocs and/or data types containing the revised type must also obtain a new version. This practice is exactly analogous to the propagation of engineering revisions from component item numbers through parent assemblies all the way up to the end product in an engineering or manufacturing bill of material. When a component part has its version incremented, so do all its parent items.

Example A `maintainSalesOrder` version eB_1 QDoc might contain version eB_1 of the two data types `SalesOrderType` and `SalesOrderLineType`. If fields specific to eB2 are added to `SalesOrderLineType` that affect the QDoc, a version eB2_1 of `SalesOrderLineType` will be created in XML schema file `SalesOrderLineType-eB2_1.xsd`. This new file will then be included in the new QDoc XML schema file `maintainSalesOrder-eB2_1.xsd`.

Moreover, if a QDoc request schema is revised resulting in a new version, a new version of the matching QDoc response schema is also created whether or not it has actually been changed, and vice versa. Thus, any QDoc response always has the same version as its corresponding QDoc request.

QDoc XML Elements

This section describes how application data is represented in QDoc XML. The selected approach complies with the SOAP 1.0 and 1.1 encoding standards in order to allow the QDocs to be exposed as remote procedure call (RPC) Web services in later releases.

Business Content Expressed as Elements

Following published guidelines within the XML standard and common practice, all application or business data within a QDoc is expressed as XML elements rather than attributes. Attributes are intended to represent metadata only—characteristics describing how the data in the XML document should be interpreted or processed, rather than application content. Moreover, the XML

schema standard on which QAD depends for syntactic validation of QDocs provides much richer typing and extensibility features (for example, ability to inherit data type characteristics by extension or by restriction) for elements than for attributes.

Denormalized Data Associations

Many business objects require the representation of data relationships between multiple entities, not only a single entity. The most common such case is the header-detail pattern, where the lines are children with a many-to-one relationship to their parent header. This pattern is often extended to more than two levels; for example, sales order headers, lines, delivery schedules, taxes, and other special charges.

Such data can be represented in the form of normalized relational tables, with all associations between the entities embodied in their use of common keys; for example, an order number in the sales order line that references the associated header. It also possible to model relationships in a structured, unnormalized manner, where the child entities are explicitly contained by their respective parents with no need for common key fields.

Following common XML usage, QDocs represent parent-child relationships between entities in a structured manner, by making each instance of a child entity a component element inside a parent element. For example, a sales order is represented by a single sales order element that contains zero or more line elements, each of which in turn may contain other types of child elements.

Common Response Data

As part of every outbound QDoc response, QDocs report the outcome of a request along with any exception conditions. Because of its inherent commonality, a single set of XML elements is specified for all QAD products.

Common response data in QDocs is handled differently depending on the version of the QDoc syntax specification in use.

QDoc Specification 1.1

One required element, `result`, is included in all QDoc response documents as a single-level child under the root. This is a QAD-defined element used to contain the result.

The `dsExceptions` element occurs exactly once in response QDocs, and contains a variable number of `temp_err_msg` elements, which in turn contain the `tt_msg_*` fields. The `exception` node used in the QDoc syntax version 1.0 translates to the `temp_err_msg` node.

Each `temp_err_msg` includes the following elements:

- `tt_msg_nbr`: identifies the type of exception (required)
- `tt_msg_desc`: briefly describes the exception condition (required)
- `tt_msg_sev`: designates the severity of the exception—informational, warning, error (optional, default error)
- `tt_msg_field`: identifies the specific field in the request for which the exception was raised (optional, no default)

- `tt_msg_context`: identifies the context within the request, typically an element or row instance, in which the exception was raised (optional, no default)
- `tt_msg_data`: contains free-form trace information about the exception; this temp-table is also used inside QAD Enterprise Applications

  Note OpenEdge errors trapped by the Service Interface are stored in this table along with application exceptions. OpenEdge errors can be identified by the OE- prefix prepended to the message number in the `tt_msg_nbr` field.
- `tt_msg_keys`: identifies application key values identifying the row associated with the exception
- `tt_msg_datetime`: identifies the date and time when the exception was raised
- `tt_msg_processed`: identifies whether or not the exception has been handled
- `tt_level`: identifies the level of the program stack in which the exception occurred
- `tt_msg_index`: identifies the unique sequential key within the table

QDoc Specification 1.0

Two required elements are included in all QDoc response documents as single-level children under the root:

- `result`, a SOAP-standard element defined in the namespace http://www.w3.org/2002/12/soap-rpc
- `returnValue`, a QAD-defined element returning the value success, warning, or error

This is pessimistic rather than optimistic usage. In the case of a multi-part QDoc (for example, a `maintainSalesOrder` request that includes two sales orders), the `returnValue` element is set to error if any of the lines in either sales order fail for any reason.

The result element always has the value `returnValue`, as its only purpose within the SOAP 1.0 standard is to reference the element within the SOAP response that contains the return value of the QDoc. This indirect approach permits the standard `result` element to be completely defined within the soap-rpc namespace, while still permitting the application to return its own strongly typed value.

Zero or more `exception` elements are included in all QDoc response documents as single-level children under the root. Each `exception` includes the following elements of the type string:

- `number`: identifies the type of exception (required)
- `description`: briefly describes the exception condition (required)
- `severity`: designates the severity of the exception
- `field`: identifies the specific field in the request for which the exception was raised (optional, no default)
- `context`: identifies the context within the request, typically an element or row instance, in which the exception was raised (optional, no default)
- `trace`: a detailed trace of the method execution that raised the exception, typically a Java stack trace (optional, no default)

It is possible for a QDoc response to contain nothing inside the QDoc root element except one or more `exception` elements plus the `result` and `returnValue` elements. This would be the case for errors recognized before application processing of the QDoc has begun; for example, if the QDoc request's name-version is not supported or its XML content is not well formed.

A common `ExceptionType` for the `exception` element is defined and stored in the QDoc namespace and common XML schema files. See "QDoc Namespaces" on page 277. Its XML schema definition is shown in the following code sample.

```
<complexType name="ExceptionType">
    <annotation>
        <documentation>
        An exception message returned from a QDoc request by
        the QAD application.
        </documentation>
    </annotation>
    <sequence>
        <!-- number and description fields are required -->
        <element name="number" type="string"/>
        <element name="description" type="string"/>
        <element name="severity" minOccurs="0"
            default="error">
            <simpleType>
               <restriction base="string">
                   <enumeration value="informational"/>
                   <enumeration value="warning"/>
                   <enumeration value="error"/>
               </restriction>
            </simpleType>
        </element>
        <element name="field" type="string" minOccurs="0"/>
        <element name="context" type="string" minOccurs="0"/>
        <element name="trace" type="string" minOccurs="0"/>
    </sequence>
</complexType>
```

A common `ReturnValueType` for the `returnValue` element is also defined and stored in the QDoc namespace and common XML schema files. Its XML schema definition is shown in the following code.

```
<simpleType name="ReturnValueType">
    <annotation>
        <documentation>
        The value returned by the QAD application.
        </documentation>
    </annotation>
    <restriction base="string">
        <enumeration value="success"/>
        <enumeration value="warning"/>
        <enumeration value="error"/>
    </restriction>
</simpleType>
```

Errors Prior to Processing

Some QDoc requests cannot be processed due to syntax errors. For example, the QDoc may not be recognized because of a misspelled name; critical attributes of the QDoc required for application processing may be invalid; or the XML may be badly formed. In such cases, a normal response based on standard naming conventions is not suitable, as the identity and contents of the QDoc are being called into question.

Since QDocs are a SOAP-compliant solution, QDoc preprocessing errors are thrown back to the sender as SOAP faults. See "SOAP Faults" on page 311.

Primitive Data Type Representations

All primitive data types used by QAD applications (character or string, integer, decimal, date, time, Boolean or logical) are represented in QDocs using the data type standards specified for XML schemas. The most interesting cases in this regard are dates, times, integers, and decimals.

- Dates are represented as YYYY-MM-DD (for example, 2003-04-22).
- Times are represented as HH:MM:SS+HH:MM (for example, 23:20:06+07:00 or 23:20:06-02:00), where the expression +HH:MM is the number of hours and minutes ahead or behind Universal Time (also called Greenwich Mean Time).
- The type dateTime is an XML-schema-defined type that is a combination of the date and time types as specified by ISO 8601. Its representation is the concatenated values of date and time separated by an upper-case T, in the format YYYY-MM-DDTHH:MM:SS+HH:MM. For example, 2003-04-22T23:20:06-08:00 would denote 11:20 pm Pacific Standard Time on April 22, 2003.
- Integers are represented normally, with an optional + or – sign preceding the digits.
- Decimals are represented with the period (.) as the decimal point and optionally the comma (,) as the thousands separator.

The application is responsible for managing the display formats used in different geographies; for example:

- Reversing the usage of the period and comma for quantities
- Modifying the sequence of the day-month-year components of dates

Default, Empty, and Null Values

As a general rule, optional elements can be omitted from any QDoc if the requestor wants the target QAD application to use their default values. A particularly important special case of this rule is the modify or change operation of a CRUD request. If an element in a modify/change request is omitted, the QAD application presumes that the element's value should not be changed and retains its existing value. This approach has the major advantage of conciseness, as it permits the requestor to populate the QDoc request with only the content that is relevant and omit the rest.

However, this approach does raise the problem of ambiguity with respect to empty values. If an omitted element signifies that its default value should be used, how do you communicate a request inside a QDoc to blank out the element's value? In other words, how should an empty value be represented? The term empty instead of null is used in order to avoid confusion with the concept of a special null object/value, such as Java null or the Progress unknown value. Unlike the null value, empty values vary based on data type. For character fields empty would be a null string, for numbers the value zero, and for logicals the value false.

To distinguish between default and empty values in QDoc requests, a distinction is made in all QDocs between an omitted element and an empty-valued one. If an element is omitted, the QAD application uses its default value. If it is empty valued (that is, has the form `<element></element>` or `<element/>`), the application assumes that the element should be assigned the empty value appropriate for its type.

While XML schema allows null elements to be explicitly represented with a nil attribute, enabled by a nillable attribute in the respective elements' schema definition, use of this construct, and null values in general, is not supported by the QDoc schemas as a general rule. However, the nil and nillable attributes may be designated for use in special cases where QAD application processing requires null value input.

Array Representation

The use of arrays in API methods and interoperability documents is generally not recommended, as they can be problematic to translate into some programming languages and often reflect a poor, unnormalized data model. Nevertheless, their use is sometimes necessary to preserve compatibility with legacy application code, if for no other reason. For such cases, you can use a standard approach for representing arrays inside QDocs.

Arrays are represented according to the SOAP encoding scheme in order to better support the automatic generation of Java client stubs from the WSDL files using non-QAD tools. Various QDoc-specific limitations are imposed in order to simplify the SOAP syntax by eliminating those features not required by any QAD applications.

QDoc arrays are elements that contain multiple occurrences of only one simple element. The element for the array as a whole is named according to the same conventions as other elements in the QDoc, based on QAD application usage.

Arrays of complex elements, multi-dimensional arrays, and arrays of arrays (jagged arrays) are not allowed in QDocs.

The approach used to represent arrays in QDocs differs depending on which QDoc specification is being used.

QDoc Specification 1.1

In the QDoc 1.1 specification the array structure has been modified to improve the ability of applications from vendors such as Progress to recognize the syntax for arrays.

The array structure in QDoc specification version 1.1 has multiple entries of the node if it is an array. Indexing is achieved by the order in which it appears in the XML.

Note In the QDoc 1.1 specification, every entry of an array must be included in a QDoc in order to pass values. Partial or "sparse" arrays are not allowed. QDoc 1.0, which followed SOAP encoding rules, did not impose this constraint.

The QDoc 1.0 specification used the following structure for arrays:

```
<soSlspsn enc:arraySize="4">
    <entry index="1">CDO</entry>
    <entry index="2">GDB</entry>
    <entry index="3">HDA</entry>
    <entry index="4">JP</entry>
</soSlspsn>
```

The equivalent array in the QDoc 1.1 specification would look like this:

```
<soSlspsn>CDO</soSlspsn>
<soSlspsn>GDB</soSlspsn>
<soSlspsn>HDA</soSlspsn>
<soSlspsn>JP</soSlspsn>
```

Note The handling of arrays was updated in QDoc 1.1 to adopt literal encoding in place of the SOAP encoding standard used in QDoc 1.0. The reason for this was that SOAP encoding imposed extra complexity on XML schemas and WSDLs, and was not widely supported by XML tools in the industry.

QDoc Specification 1.0

In QDoc specification 1.0 the single component element is named `entry` in all QDocs regardless of usage, with `entry` defined in the QDoc common namespace so as not to conflict with the name of any QAD application array. Every array described in a QDoc version 1.0 XML schema is declared as a restriction of the base type `enc:Array`. Every QDoc instance contains `itemType` and `arraySize` attributes associated with the following SOAP encoding namespace URI:

http://www.w3.org/2002/12/soap-encoding

The value of `itemType` can be any one of the following: string, Boolean, decimal, int, date, time, anyType. (anyType is the theoretical supertype of all other types, somewhat like Object in Java.) The value of `arraySize` is an integer denoting the maximum size or extent of the array, or the asterisk character (*) if that value is not known.

Because SOAP 1.0 requires that arrays be included in their entirety (that is, no partial arrays), a means is needed to distinguish an array element that is being set to an empty value by a QDoc from one that is not being changed by the QDoc.

Note This is the same problem of specifying empty versus default values, except that in the case of arrays it is not possible to skip an empty element in order to designate that the default value should be used. See “Default, Empty, and Null Values” on page 282.

In order to resolve the problem, the QDoc-defined `entry` element has one required attribute `index` and one optional attribute `skip`, also defined in the QDoc common namespace. The `index` attribute identifies each entry of the array. The `skip` attribute indicates whether the entry should be skipped or defaulted if set to true. If the `skip` attribute is set to true, the content of its containing `entry` element is ignored.

Syntactically, the indexed entries of the array can appear in any order. However, by convention, they should always be sequenced in ascending index order. For example, the following QDoc fragment would represent a six-member numeric array called `orderQuantity` containing two empty, two skipped, and two non-empty elements.

```
<qdoc:orderQuantity
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:enc="http://www.w3.org/2002/12/soap-encoding"
    xmlns:qcom="http://www.qad.com/qdoc/common"
    enc:itemType="xsi:decimal"
    enc:arraySize="6"
    >
    <!-- Entries 1, 3 are set to empty and 4, 6 are skipped -->
    <qcom:entry index="1"/>
    <qcom:entry index="2">10.583</qcom:entry>
    <qcom:entry index="3"/>
    <qcom:entry index="4" skip="true"/>
    <qcom:entry index="5">22</qcom:entry>
    <qcom:entry index="6" skip="true">ignore!</qcom:entry>
</qdoc:orderQuantity>
```

The following is the XML schema definition for this fragment, defined using SOAP conventions:

```
<schema
    xmlns="http://www.w3.org/2001/XMLSchema"
```

```
    xmlns:enc="http://www.w3.org/2002/12/soap-encoding"
    xmlns:qcom="http://www.qad.com/qdoc/common"
    xmlns:qdoc="http://www.qad.com/qdoc/eb">
    targetNamespace="http://www.qad.com/qdoc/eb"
    xml:lang="EN"
    >
    <import namespace="http://www.w3.org/2002/12/soap-encoding"
        schemaLocation="soap-encoding.xsd"/>
    <element name="orderQuantity">
        <complexType>
            <complexContent>
               <restriction base="enc:Array">
                   <sequence>
                       <element ref="qcom:entry"
                           minOccurs="6" maxOccurs="6"/>
                   </sequence>
                   <attributeGroup ref="enc:arrayAttributes"/>
                   <attributeGroup ref="enc:commonAttributes"/>
               </restriction>
            </complexContent>
        </complexType>
    </element>
</schema>
```

The following is the XML schema fragment defining the `entry` element and its `index` and `skip` attributes in the QDoc common namespace:

```
<schema
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:qcom="http://www.qad.com/qdoc/common"
    targetNamespace="http://www.qad.com/qdoc/common"
    xml:lang="EN"
    >
    <element name="entry">
        <complexType>
            <simpleContent>
               <extension base="anyType">
                   <attribute name="index" type="positiveInteger"
                       use="required"/>
                   <attribute name="skip" type="boolean"/>
               </extension>
            </simpleContent>

        </complexType>
    </element>
</schema>
```

Element Names

Except for the single root element that identifies the QDoc, a universal naming convention for QDoc elements that applies to all QAD products does not exist. Rather, product-specific conventions are used. See “Array Representation” on page 283.

QDoc XML Attributes

All QDocs using the 1.0 syntax specification use the XML attributes described in this section.

Note Several items that were attributes in the QDoc 1.0 specification and described in this section—`version`, `mnemonicsRaw`, `scopeTransaction`—have been moved to the `dsSessionContext` structure in the QDoc 1.1 specification and are now elements. For details see “Session Context” on page 268. In addition, in the QDoc 1.1 specification the `suppressResponseDetail` attribute is a node in the `ReferenceParameters` section.

xml:lang

This attribute is an XML standard, not QDoc-specific. It designates the national language of its associated element, using the two-letter language codes defined by the ISO 639 standard.

Note This attribute is not used in the QDoc 1.1 specification.

version

This attribute is specific to QDocs. It identifies the version of a QDoc schema to which a particular instance of that QDoc conforms, and is required at the QDoc root element level. Its format is `<major product release>_<sequence>`, where `<major product release>` is the lowest major release of the QAD application product in which the QDoc is supported (for example, eB, eB2, QAD SE, or above for QAD EE), and `<sequence>` is an incremented sequence number within that release starting with 1. Any periods in the release name are removed.

Example The first version of an QAD Enterprise Applications QDoc that is developed to work with both the eB and eB2 releases would have the version eB_1, for both eB and eB2 users. If this QDoc were later enhanced only for releases QAD SE and above, the new version would be called ERP3_1, and could coexist with version eB_1 if QAD decides to support both versions concurrently. If the same enhancement were later retrofit to eB and eB2, the new version for the eB and eB2 releases would be called eB_2. Note that service packs are not treated as major releases for QDoc versioning purposes.

This versioning scheme allows multiple versions of the same QDoc to be supported concurrently for multiple product releases, while capturing in a natural way the relationships and compatibilities among the various QDoc versions and product releases.

For custom or user-extended QDocs, QAD suggests that custom version designators be assigned as suffixes to the QAD-supported versions on which they are based (for example, eB_2_ABC_1 for the first customization developed by ABC Company). While all customized QDocs must be defined in a namespace different from QAD's in order to avoid name-version clashes, customer-specific version identifiers help highlight their custom nature. See "QDoc Namespaces" on page 277.

mnemonicsRaw

This attribute is an optional Boolean for QDoc requests at the root element level that indicates whether during application processing the QDoc should send the language-specific mnemonic (if the value equals False), or whether the numeric code should be sent instead (value equals True).

Some options in QAD Enterprise Applications functions appear on screen using alphabetic codes or words. Internally, these options are controlled by numeric codes. You can change, add, and delete the mnemonics and labels associated with these codes using Language Detail Maintenance (36.4.3).

Note In the QDoc 1.1 specification this attribute has been moved to `dsSessionContext` and is an element. For details see "Session Context" on page 268.

scopeTransaction

This attribute is an optional Boolean for QDoc requests at the root element level that indicates whether the processing of the QDoc should be scoped as a single transaction. In general, units of work scoped by the application programs that process QDocs vary based on the nature of the request, the input data, and decisions made by the application designers. The `scopeTransaction` attribute allows the requestor to override normal units of work and force a single-transaction scope for the request.

In distributed environments where the requesting and receiving application are running in different application server containers or different machines, controlling the transaction scope can be inconvenient or even impossible for the requestor of a remote service. This attribute provides a simple means of controlling the transaction scope remotely.

Note In the QDoc 1.1 specification this attribute has been moved to `dsSessionContext` and is an element.

logTransaction

This attribute is an optional Boolean for QDoc requests at the root element level that indicates whether during application processing the QDoc should log the request internally for audit trail or recovery purposes. Because of the potential performance overhead associated with logging, requestors may want to avoid the cost of logging, particular in local integrations running in a stable, local network environment.

Logging is most important in distributed environments, where the QDoc application may be deployed across multiple application servers or machines using communication links that are less than 100% reliable. Even when an integration framework is receiving all QDoc requests, logging at this level may not be sufficient when the application is running as multiple components in separate environments. It may be necessary for the application to write log entries in order to protect against internal communication failures between the application component and the integration framework.

Note Because there is no equivalent of the `logtransaction` attribute in the QDoc 1.1 syntax specification, `logtransaction` has been removed from that version of the syntax.

transactionID

This attribute is an optional string for QDoc requests at the root element level that designates a requestor-defined identifier. It is used only in conjunction with the `logTransaction` attribute. It is entirely requestor-defined, and does not have to be a unique key. The intent is to capture its value in any QDoc log entries, so that remote requestors can refer to an audit trail entry in case of internal communication failures.

Note Because there is no equivalent of the `transactionID` attribute in the QDoc 1.1 syntax specification, `transactionID` has been removed from that version of the syntax.

suppressResponseDetail

This attribute is an optional string for QDoc requests at the root element level that indicates whether the business data of the QDoc response is needed, as opposed to only the exceptions raised during processing. It is provided as a performance-enhancing feature, when the requestor wants a request processed but does not need to know the results outside of the overall outcome and any exceptions encountered.

A primary example would be the processing of inbound QDocs into QAD Enterprise Applications through Q/LinQ or QAD EDI ECommerce. As both of these products perform their transaction processing asynchronously, they need only to log success/failure and exceptions thrown, from which they may (or may not) create separate outbound QDoc confirmation documents for the ultimate requestor. Given the verbose nature of XML in general, using this attribute could measurably improve performance and/or conserve network bandwidth.

Query requests would always set this attribute to false; otherwise, the QDoc response would not include the results of the query.

Note In the QDoc 1.1 specification the `suppressResponseDetail` attribute is a node in the SOAP header `ReferenceParameters` section:

```
<ns0:ReferenceParameters xmlns:ns0="http://www.w3.org/2005/08/addressing">
    <suppressResponseDetail xmlns="urn:schemas-qad-com:xml-services:common">
    true</suppressResponseDetail>
</ns0:ReferenceParameters>
```

Common Attribute Group

In the QDoc 1.0 specification several attributes—`version, mnemonicsRaw, scopeTransaction, logTransaction, transactionId`, and `suppressResponseDetail`—are used in almost all QDocs. Hence, they are defined once as members of an XML attribute group defined in common XML schema files, and referenced in all relevant QDoc definitions. See “QDoc Namespaces” on page 277.

Note In the QDoc 1.1 specification several of these attributes have been moved to either the `dsSessionContext` structure or the SOAP header `ReferenceParameters` section; consequently there is no common attribute group in the QDoc 1.1 specification.

The following is the XML schema definition fragment for this group.

```
<import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="xml.xsd"/>
<attributeGroup name="qcom:commonAttributes">
    <attribute name="version" type="string"
        use="required"/>
    <attribute name="mnemonicsRaw" type="boolean"
        use="optional"/>
    <attribute name="scopeTransaction" type="boolean"
        use="optional"/>
    <attribute name="logTransaction" type="boolean"
        use="optional"/>
    <attribute name="transactionId" type="string"
        use="optional"/>
    <attribute name="suppressResponseDetail" type="boolean"
        use="optional"/>
    <attribute ref="xml:lang"/>
</attributeGroup>
```

QAD Enterprise Applications-Specific QDoc Syntax

The XML-based content of all QAD Enterprise Applications QDoc requests and responses is mapped or bound at run-time to and from the native data structures of the UI, service interface, or QAD JIT Sequencing API responsible for processing. This section summarizes the fixed rules used to accomplish this binding.

If a data structure clash or mismatch is detected between the QDoc syntax and the service interface and QAD JIT Sequencing API signatures, a QDoc syntax error is thrown to the requestor and the request is not processed.

QDoc Name

The QDoc name or root element maps to the name of the API method to be run, in the case of the service interface and QAD JIT Sequencing APIs. In the case of the UI APIs, it is used to determine the QAD Enterprise Applications online program to run.

Simple Elements

Simple QDoc elements map to temp-table fields in the case of service interface and QAD JIT Sequencing APIs, or user input fields in the case of the UI APIs. Their names match the QAD Enterprise Applications names of the corresponding service interface and QAD JIT Sequencing API parameters, except that mixed-case or camel-case notation for Java classes is used in place of the underscore-delimited names common in QAD Enterprise Applications. That is, the underscore characters separating tokens within the QAD Enterprise Applications name are removed and the first letter of each token within the name is capitalized, while all other letters as well as the first letter are lower-case. For example, the QAD Enterprise Applications field pt_prod_line would be expressed as `ptProdLine` in the QDocs.

All arrays are presumed to contain simple elements. Each is mapped to a temp-table field with extent equal to the array's declared length.

Enumerated fields used across many QDocs (for example, the operation field used for CRUD processing in the service interface and QAD JIT Sequencing APIs) are defined in the common QDoc XML schema file for inclusion in all schemas.

Complex Elements

Complex QDoc elements containing other elements, representing business data entities in QAD Enterprise Applications, map to iterating frames in the case of the UI APIs and temp-tables in the case of service interface and QAD JIT Sequencing APIs. Their names are natural business terms expressed in camel-case with no database-specific notation (for example, pt_mstr for item) or special prefixes/suffixes. For elements that are associated with temp-table parameters in the service interface and QAD JIT Sequencing API methods, the QDoc element name is identical to the temp-table name but without the `tt` prefix and with the first letter changed to lower-case. For example, the temp-table `ttPurchaseOrderDet` in the `maintainPurchaseOrder` API method would be expressed as `purchaseOrderDet` in the QDocs.

Just as the service interface and QAD JIT Sequencing API temp-tables make heavy use of common field lists (for example, the include files `mfaittxt.i`, `mfctit01.i`, and so on), so do their corresponding QDoc elements. XML schema supports an inheritance mechanism whereby types can be derived by extending other types by adding fields, analogous to subclassing in object-oriented programming languages.

Using this technique, an abstract base type `ApiTempTableType` is defined that includes all the fields common to all service interface and QAD JIT Sequencing API temp-tables. Reusable derived types such as `TransCommentType` are built from `ApiTempTableType` by extension, and in turn used to build more specific types such as `PurchaseOrderCmtType`. The type hierarchy is built by inspecting the present service interface and QAD JIT Sequencing API temp-table definitions and common field lists. The collections of common data are modeled as base types in the XML schema.

The following XML schema definition fragment is for the `ApiTempTableType`, as well as fragments of the `TransCommentType`, `PurchaseOrderCmtType`, and `OperationType` definitions. The `ApiTempTableType` definition is stored in the XML schema file for common QAD Enterprise Applications QDoc objects, while the derived types are stored in separate schema files. See “QDoc Namespaces” on page 277.

```
<complexType name="ApiTempTableType" abstract="true">
    <sequence>
        <!-- Common API temp-table fields from mfaittxt.i -->
        <element name="apiSequence" type="string"/>
        <element name="apiSuccess" type="boolean"/>
        <element name="apiExternalKey" type="string"/>
    </sequence>
</complexType>

<simpleType name="OperationType">
    <restriction base="string">
        <!-- Common Operation values from mfaiocon.i -->
        <enumeration value="A"/> <!-- Add -->
        <enumeration value="M"/> <!-- Modify -->
        <enumeration value="R"/> <!-- Remove -->
        <enumeration value="U"/> <!-- Unmodified -->
        <enumeration value="S"/> <!-- Sync -->
        <enumeration value="G"/> <!-- Get -->
        <enumeration value="N"/> <!-- No-op -->
    </restriction>
</simpleType>

<complexType name="TransCommentType">
    <complexContent>
        <extension base="qdoc:ApiTempTableType">
            <sequence>
               <!-- Common Transaction Comment temp-table fields
                   from mfctit01.i, mfctid01.i -->
               <element name="operation" type="qdoc:OperationType"
                   minOccurs="0"/>
               <element name="seq" type="int" minOccurs="0"/>
               <element name="ref" type="string" minOccurs="0"/>
               <element name="type" type="string" minOccurs="0"/>
               <element name="lang" type="string" minOccurs="0"/>
               ...
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="PurchaseOrderCmtType">
    <complexContent>
        <!-- No P.O.-specific fields to add to base Transaction
            Comments, but the type is still required in order
            to map easily to the corresponding API temp-table. -->
        <extension base="qdoc:TransCommentType"/>
```

```
    </complexContent>
</complexType>
```

Note QDoc schemas in the 1.1 standard use a flat structure—there is no superschema.

Arrays

Arrays are mapped to QAD Enterprise Applications fields with extents greater than 0. Only one-dimensional arrays that contain a single simple element type are allowed in QDocs. The name of the array maps to the QAD Enterprise Applications field name.

In the QDoc 1.1 specification the field name is repeated in sequential order in the XML to provide the indexing.

In the QDoc 1.0 specification the name of all array elements is `entry`, defined in the QDoc common namespace.

For details, see “Array Representation” on page 283.

Element Order

In general, an XML schema allows the child elements under a given parent to be defined such that the element order is variable. Use of this feature would support QDoc syntax flexibility. However, it is not used for QAD Enterprise Applications QDocs because of the following constraints:

- All the complex elements corresponding to service interface and QAD JIT Sequencing API temp-tables are derived by extending the abstract base type `ApiTempTable`, in order to define all the standard boilerplate temp-table fields in one place. However, the current release of XML schema forces all fields defined in the base type to precede the fields defined in the derived type, thus constraining the element order. See “Complex Elements” on page 289.
- The QDocs intended for service interface and QAD JIT Sequencing API processing contain primarily multi-occurrence, complex child elements corresponding to the service interface and QAD JIT Sequencing API temp-tables. However, use of variable element order underneath a given parent is allowed only if all the children are single-occurrence elements.

Required vs. Optional Elements

In general, XML schema allows the child elements under a given parent to be defined as required or optional through the `minOccurs` and `maxOccurs` attributes. The default is `minOccurs=1, maxOccurs=1`.

However, to make maintenance of the XML schema easier over time, virtually all QDoc elements are defined as optional (`minOccurs=0`). This approach also supports many common API scenarios in which a common data object is used in multiple API methods, but with context-specific validation that determines whether the elements are required or optional. It is the responsibility of the QAD Enterprise Applications application to throw exceptions if the requestor does not provide a required field using present QAD Enterprise Applications validation logic, rather than relying on XML schema syntax checking.

Normalized vs. Denormalized Representation

In the case of service interface and QAD JIT Sequencing APIs, all temp-table parameters are expressed in normalized form, in which the relationship between parent and child temp-tables is represented only as a set of common key fields (foreign keys) that occur in both tables. That is, one temp-table is not contained in or structurally subordinate to the other. Rather, all appear as siblings in the API signature.

A complex QDoc element that contains another complex QDoc element (in denormalized fashion) is mapped to two separate temp-tables in the service interface and QAD JIT Sequencing APIs. In addition, all designated key fields appearing in the parent element are duplicated inside the child temp-table records with the same values as in their respective parent temp-table records. In other words, the normalized temp-table records contain foreign key values referencing their respective temp-table parent records, as with any normalized relational data structure.

In order to designate those simple QDoc elements (fields) inside a particular complex QDoc element (temp-tables) that constitutes the primary key, a special `primaryKeys` attribute is defined in the QDoc namespace for inclusion in all QDoc XML schema files. Its definition is stored in common XML schema files.

This attribute is used with all XML schema element definitions that are bound to service interface and QAD JIT Sequencing API temp-tables, so that the QDoc Request Handler can identify the parent-child and key relationships from the XML schema and properly normalize the resulting temp-tables. The following QDoc examples illustrate use of this attribute within QDocs.

The XML schema definition for the `primaryKeys` attribute is as follows.

```
<attribute name="primaryKeys" type="token">
    <annotation>
        <documentation>
        Lists the primary keys of the QDoc element for the
        purpose of normalizing its child elements into sibling
        data structures with fully unique keys.
        Used to map the unnormalized QDoc XML into a temp-table
        representation for the service interface APIs.
        </documentation>
    </annotation>
</attribute>
```

The UI APIs work natively with denormalized data structures, and do not require such transformation to normalize the QDoc elements.

CRUD QDocs

QDocs that request basic create-read-update-delete (CRUD) maintenance against some QAD Enterprise Applications business object are very common. Because of their ubiquity, a single format is used for all of them along the lines of the signature standards defined for the service interface and QAD JIT Sequencing APIs.

- All QDocs requesting CRUD activity on an QAD Enterprise Applications entity are named using the verb maintain (for example, `maintainSalesOrder`, `maintainSupplier`). This terminology keeps the QDoc names relatively close to the QAD Enterprise Applications menu procedures whose functionality they incorporate.

- At the beginning of every complex XML element that denotes a business object to be maintained (for example item, `salesOrder`, `supplier`), an operation element describes the type of maintenance requested for the object. Allowed values are A (add), M (modify), D (delete), G (get), S (sync), N (no-op).
  - Sync indicates that the object should be added if not present in the database and modified if present, essentially the same way that QAD Enterprise Applications maintenance procedures currently work.
  - Add raises an error if the object already exists.
  - Modify raises an error if the object does not exist.
  - No-op is a special value signifying that the element should be skipped; it may be useful for cases in which the value of some QDoc element helps navigate to or access subsequent elements, but for which no processing is required.

Operation elements can exist at multiple levels within a QDoc, such as the order header, line, and transaction comment levels.

Attributes

For QDocs in the 1.0 specification that are implemented by service interface and QAD JIT Sequencing APIs, the `scopeTransaction`, `logTransaction`, and `transactionId` attributes are mapped to the standard `scopeTrans`, `logTrans`, and `transId` parameters respectively. The `suppressResponseDetail` attribute has no counterpart in the service interface and QAD JIT Sequencing API call signatures.

For QDocs in the 1.1 specification, the `scopeTransaction` attribute has been moved to the `dsSessionContext` structure, and both the `logTransaction` and `transactionId` attributes have been removed.

QDoc Extensions and Customizations

Because bindings between QDocs and QAD Enterprise Applications are based on a fixed set of rules, the only way to extend or modify a QDoc is to extend or modify:

- The QAD Enterprise Applications code that implements the QDoc, and
- The XML schema for the QDoc

The new/modified QDoc elements must follow the naming and format conventions described for standard QDoc content, but must be defined within a non-QAD namespace to distinguish them from QAD-standard syntax.

If the custom QDoc is to be processed using the QAD Enterprise Applications menu procedures (the UI APIs), an XML schema for the QDoc should be generated using the QGen tool. See Chapter 18, “QGen,” on page 215.

QDoc Examples

This section contains examples of both the QDoc 1.1 specification and the 1.0 specification as indicated.

QAD Enterprise Applications Inbound QDoc Request (1.1)

This is a sample QDoc request to maintain a sales order in QAD Enterprise Applications, based on the API method `maintainSalesOrder`. The QDoc message envelope is omitted, as it is described elsewhere. See “QDoc Message Envelope” on page 301.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Body>
    <maintainSalesOrder>
        <qcom:dsSessionContext>
            <qcom:ttContext>
                <qcom:propertyQualifier>QAD</qcom:propertyQualifier>
                <qcom:propertyName>domain</qcom:propertyName>
                <qcom:propertyValue />
            </qcom:ttContext>
            <qcom:ttContext>
                <qcom:propertyQualifier>QAD</qcom:propertyQualifier>
                <qcom:propertyName>scopeTransaction</qcom:propertyName>
                <qcom:propertyValue>false</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:ttContext>
                <qcom:propertyQualifier>QAD</qcom:propertyQualifier>
                <qcom:propertyName>mnemonicsRaw</qcom:propertyName>
                <qcom:propertyValue>false</qcom:propertyValue>
            </qcom:ttContext>
            <qcom:ttContext>
                <qcom:propertyQualifier>QAD</qcom:propertyQualifier>
                <qcom:propertyName>username</qcom:propertyName>
                <qcom:propertyValue />
            </qcom:ttContext>
            <qcom:ttContext>
                <qcom:propertyQualifier>QAD</qcom:propertyQualifier>
                <qcom:propertyName>password</qcom:propertyName>
                <qcom:propertyValue />
            </qcom:ttContext>
            </qcom:dsSessionContext>
        <dsSalesOrder>
            <salesOrder>
                <operation>text</operation>
                <soNbr>SO987654</soNbr>
                <soCust>001</soCust>
                <soSlspsn>CDO</soSlspsn>
                <soSlspsn>GDB</soSlspsn>
                <soSlspsn>HDA</soSlspsn>
                <soSlspsn>JP</soSlspsn>
                <multSlspsn>yes</multSlspsn>
                <salesOrderTransComment>
                    <operation>text</operation>
                    <cmtCmmt>Line 1</cmtCmmt>
                    <cmtCmmt/>
                    <cmtCmmt/>
                    <cmtCmmt/>
                    <cmtCmmt/>
                    <cmtCmmt/>
                    <cmtCmmt/>
                    <cmtCmmt/>
                    <cmtCmmt/>
                    <cmtCmmt>Line 10</cmtCmmt>
                </salesOrderTransComment>
                <salesOrderDetail>
                    <operation>text</operation>
                    <line>1</line>
                    <sodPart>item6</sodPart>
                    <sodQtyOrd>1</sodQtyOrd>
                </salesOrderDetail>
```

```
                <salesOrderDetail>
                    <operation>text</operation>
                    <line>2</line>
                    <sodPart>10-15000</sodPart>
                    <sodQtyOrd>1</sodQtyOrd>
                </salesOrderDetail>
            </salesOrder>
        </dsSalesOrder>
    </maintainSalesOrder>
</soapenv:Body>
```

QAD Enterprise Applications QDoc Response (1.1)

This is a sample partial QDoc response based on the API method `maintainItemMaster`. The QDoc message envelope is omitted, as it is described elsewhere. See “QDoc Message Envelope” on page 301.

```
<soapenv:Body>
    <ns1:maintainItemMasterResponse xmlns="urn:schemas-qad-com:xml-services" xmlns:ns1=
    "urn:schemas-qad-com:xml-services">
        <ns1:result>error</ns1:result>
        <ns2:dsSessionContext xmlns:ns2="urn:schemas-qad-com:xml-services:common">
            <ns2:ttContext>
                <ns2:propertyQualifier>QAD</ns2:propertyQualifier>
                <ns2:propertyName>domain</ns2:propertyName>
                <ns2:propertyValue>QP</ns2:propertyValue>
            </ns2:ttContext>
            <ns2:ttContext>
                <ns2:propertyQualifier>QAD</ns2:propertyQualifier>
                <ns2:propertyName>scopeTransaction</ns2:propertyName>
                <ns2:propertyValue>false</ns2:propertyValue>
            </ns2:ttContext>
            <ns2:ttContext>
                <ns2:propertyQualifier>QAD</ns2:propertyQualifier>
                <ns2:propertyName>version</ns2:propertyName>
                <ns2:propertyValue>eB2_2</ns2:propertyValue>
            </ns2:ttContext>
            <ns2:ttContext>
                <ns2:propertyQualifier>QAD</ns2:propertyQualifier>
                <ns2:propertyName>mnemonicsRaw</ns2:propertyName>
                <ns2:propertyValue>true</ns2:propertyValue>
            </ns2:ttContext>
            <ns2:ttContext>
                <ns2:propertyQualifier>QAD</ns2:propertyQualifier>
                <ns2:propertyName>username</ns2:propertyName>
                <ns2:propertyValue />
            </ns2:ttContext>
            <ns2:ttContext>
                <ns2:propertyQualifier>QAD</ns2:propertyQualifier>
                <ns2:propertyName>password</ns2:propertyName>
                <ns2:propertyValue />
            </ns2:ttContext>
        </ns2:dsSessionContext>
        <ns2:dsExceptions xmlns:ns2="urn:schemas-qad-com:xml-services:common">
            <ns2:temp_err_msg>
                <ns2:tt_level />
                    <ns2:tt_msg_context>itemMaster#1(ptPart-02-0001)
                    field(ptDesc1)</ns2:tt_msg_context>
                    <ns2:tt_msg_data />
                    <ns2:tt_msg_datetime>2007-08-02T10:43:23+1000</ns2:tt_msg_datetime>
                    <ns2:tt_msg_desc>Submitted data was truncated</ns2:tt_msg_desc>
                    <ns2:tt_msg_field />
                <ns2:tt_msg_index>0</ns2:tt_msg_index>
                <ns2:tt_msg_keys />
                <ns2:tt_msg_keys />
                <ns2:tt_msg_keys />
                <ns2:tt_msg_keys />
                <ns2:tt_msg_keys />
                <ns2:tt_msg_keys />
```

```
                <ns2:tt_msg_nbr>MfgProWarningMessage</ns2:tt_msg_nbr>
                <ns2:tt_msg_processed>false</ns2:tt_msg_processed>
                <ns2:tt_msg_sev>warning</ns2:tt_msg_sev>
            </ns2:temp_err_msg>
            <ns2:temp_err_msg>
                <ns2:tt_level />
                <ns2:tt_msg_context>itemMaster#1(ptPart-02-0001)</ns2:tt_msg_context>
                <ns2:tt_msg_data />
                <ns2:tt_msg_datetime>2007-08-02T10:43:23+1000</ns2:tt_msg_datetime>
                <ns2:tt_msg_desc>ERROR: INVALID SITE. Please re-enter.</ns2:tt_msg_desc>
                <ns2:tt_msg_field />
                <ns2:tt_msg_index>0</ns2:tt_msg_index>
                <ns2:tt_msg_keys />
                <ns2:tt_msg_keys />
                <ns2:tt_msg_keys />
                <ns2:tt_msg_keys />
                <ns2:tt_msg_keys />
                <ns2:tt_msg_keys />
                <ns2:tt_msg_nbr>MfgProErrorMessage</ns2:tt_msg_nbr>
                <ns2:tt_msg_processed>false</ns2:tt_msg_processed>
            <ns2:tt_msg_sev>error</ns2:tt_msg_sev>
            </ns2:temp_err_msg>
            <ns2:temp_err_msg>
                <ns2:tt_level />
                <ns2:tt_msg_context>itemMaster#1(ptPart-02-0001)
                field(ptSite)</ns2:tt_msg_context>
                <ns2:tt_msg_data />
                <ns2:tt_msg_datetime>2007-08-02T10:43:23+1000</ns2:tt_msg_datetime>
                <ns2:tt_msg_desc>Error processing data. See Context for
                details.</ns2:tt_msg_desc>
                <ns2:tt_msg_field />
                <ns2:tt_msg_index>0</ns2:tt_msg_index>
                <ns2:tt_msg_keys />
                <ns2:tt_msg_keys />
                <ns2:tt_msg_keys />
                <ns2:tt_msg_keys />
                <ns2:tt_msg_keys />
                <ns2:tt_msg_keys />
                <ns2:tt_msg_nbr>EventExceptionex001</ns2:tt_msg_nbr>
                <ns2:tt_msg_processed>false</ns2:tt_msg_processed>
                <ns2:tt_msg_sev>error</ns2:tt_msg_sev>
            </ns2:temp_err_msg>
        </ns2:dsExceptions>
        <ns1:dsItemMasterResponse>
            <ns1:itemMaster>
                <ns1:ptPart>02-0001</ns1:ptPart>
            </ns1:itemMaster>
        </ns1:dsItemMasterResponse>
    </ns1:maintainItemMasterResponse>
</soapenv:Body>
```

QAD Enterprise Applications Inbound QDoc Request (1.0)

This is a sample partial QDoc request to maintain a purchase order in QAD Enterprise Applications, based on the API method `maintainPurchaseOrder`. The QDoc message envelope is omitted, as it is described elsewhere. See “QDoc Message Envelope” on page 301.

The examples assume that the earliest supported QAD Enterprise Applications release is eB.

```
<?xml version="1.0" encoding="UTF-8"?>
<maintainPurchaseOrder version="eB_1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:enc="http://www.w3.org/2002/12/soap-encoding"
    xmlns="http://www.qad.com/qdoc/eb"
    xmlns:qcom="http://www.qad.com/qdoc/common"
    schemaLocation="http://www.qad.com/qdoc/eb/schemas"
    scopeTransaction="true"
    logTransaction="true"
    transactionId="abc123"
```

```
    suppressResponseDetail="false"
    xml:lang="EN"
    >
    <purchaseOrder>
        <operation>A</operation>
        <nbr>PO1234</nbr>
        <vend>00000001</vend>
        <ordDate>2002-04-01</ordDate>

        <rmks>This is a test PO</rmks>
        <purchaseOrderCmt>
            <operation>A</operation>
            <seq>01</seq>
            <cmmt>Line 1 of comment!</cmmt>
        </purchaseOrderCmt>
        <purchaseOrderDet>
            <operation>A</operation>
            <line>01</line>
            <dueDate>2002-05-01</dueDate>
            <part>10-10000</part>
            <qtyOrd>50</qtyOrd>
        </purchaseOrderDet>
        <purchaseOrderDet>
            <operation>A</operation>
            <line>02</line>
            <dueDate>2002-06-01</dueDate>
            <part>10-15000</part>
            <qtyOrd>20</qtyOrd>
        </purchaseOrderDet>

    </purchaseOrder>
</maintainPurchaseOrder>
```

The following XML schema describes the syntax of the preceding sample. The `PurchaseOrderType` definition is used in the QDoc request and is stored in a separate file and included in the top-level schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This schema would be stored in file
    maintainPurchaseOrder-1_0.xsd -->
<schema
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:enc="http://www.w3.org/2002/12/soap-encoding"
    xmlns:qdoc="http://www.qad.com/qdoc/eb"
    xmlns:qcom="http://www.qad.com/qdoc/common"
    targetNamespace="http://www.qad.com/qdoc/eb"
    xml:lang="EN"
    >
    <!-- Include SOAP Encoding definitions,
        common QAD Enterprise Applications QDoc fields,
        purchase order definitions. -->
    <import namespace="http://www.w3.org/2002/12/soap-encoding"
        schemaLocation="soap-encoding.xsd"/>
    <include schemaLocation="qdocCommon-eB_99.xsd"/>
    <include schemaLocation="PurchaseOrderType-eB_1.xsd"/>
    <element name="maintainPurchaseOrder">
        <complexType>
            <attributeGroup ref="qcom:commonAttributes"/>
            <sequence>
               <element name="purchaseOrder"
                   type="qdoc:PurchaseOrderType"
                   minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
        </complexType>
    </element>
</schema>
```

The following XML schema fragment defines `PurchaseOrderType` and `PurchaseOrderDetType`.

```
    <!-- This schema would be stored in file
```

```
        PurchaseOrderType.xsd -->
    <!-- Include SOAP Encoding definitions,
        common QAD Enterprise Applications QDoc fields,
        transaction comment types. -->
    <import namespace="http://www.w3.org/2002/12/soap-encoding"
        schemaLocation="soap-encoding.xsd"/>
    <import namespace="http://www.qad.com/qdoc/common"
        schemaLocation="qdocCommon-qad_1.xsd"/>
    <include schemaLocation="qdocCommon-eB_99.xsd"/>
    <include schemaLocation="TransCommentType-eB_1.xsd"/>

    <!-- PurchaseOrderType extends the common ApiTempTableType
        in order to pick up the common temp-table fields -->
    <!-- Primary key(s) of PurchaseOrderType are listed in
        qdoc:primaryKeys for use when normalizing the data -->
    <complexType name="PurchaseOrderType" qdoc:primaryKeys="nbr">
        <complexContent>
            <extension base="qdoc:ApiTempTableType">
               <sequence>
                   <!-- Note: OperationType is defined in the common
                           QDoc XML Schema file to allow the
                           enumerated Values "A" (add), "M" (modify),
                           "R" (remove), “U” (unmodified), and "S" (sync). -->
               <element name="operation"
                   type="qdoc:OperationType"
                   minOccurs="0"/>
               <element name="nbr" type="string" minOccurs="0"/>
               <element name="vend" type="string" minOccurs="0"/>
                ...
               <!-- Reuse common TransCommentType here -->
               <element name="purchaseOrderCmt"
                   type="qdoc:TransCommentType"
                   minOccurs="0" maxOccurs="unbounded"/>
               <element name="purchaseOrderDet"
                   type="qdoc:PurchaseOrderDetType"
                   minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
...

<!-- PurchaseOrderDetType extends the common
    ApiTempTableType in order to pick up the common
    temp-table fields -->
<!-- Primary key(s) of PurchaseOrderDetType are listed in
    qdoc:primaryKeys for use when normalizing the data -->
<complexType name="PurchaseOrderDetType"
    qdoc:primaryKeys="line">
    <complexContent>
        <extension base="qdoc:ApiTempTableType">
            <sequence>
               <element name="operation" type="qdoc:OperationType"
                   minOccurs="0"/>
               <element name="line" type="int" minOccurs="0"/>
               ...
            </sequence>
        </extension>
    </complexContent>
</complexType>
...
```

QAD Enterprise Applications QDoc Response (1.0)

This is a sample partial QDoc response to the preceding QDoc request based on the API method `maintainPurchaseOrderResponse`. In this example, as with most service interface and QAD JIT Sequencing API methods, the input data is echoed back as output with any changed or default values, as well as any exception messages at the bottom.

The QDoc message envelope is omitted, as it is described elsewhere. See “QDoc Message Envelope” on page 301. The examples presume that the earliest supported QAD Enterprise Applications release is eB.

```
<?xml version="1.0" encoding="UTF-8"?>
<maintainPurchaseOrderResponse version="eB_1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:enc="http://www.w3.org/2002/12/soap-encoding"
    xmlns:rpc="http://www.w3.org/2002/12/soap-rpc"
    xmlns="http://www.qad.com/qdoc/eb"
    xmlns:qcom="http://www.qad.com/qdoc/common"
    schemaLocation="http://www.qad.com/qdoc/eb/schemas"
    xml:lang="EN"
    >
    <rpc:result>returnValue</rpc:result>
    <returnValue>success</returnValue>
    <purchaseOrder>
        <nbr>PO1234</nbr>
        <vend>00000001</vend>
        <ordDate>2002-04-01</ordDate>
        <rmks>This is a test PO</rmks>
        ...
        <purchaseOrderCmt>
            <seq>01</seq>
            ...
            <cmmt>Line 1 of comment!</cmmt>
        </purchaseOrderCmt>
        <purchaseOrderDet>
            <line>01</line>
            <dueDate>2002-05-01</dueDate>
            <part>10-10000</part>
            <qtyOrd>50</qtyOrd>
            ...
        </purchaseOrderDet>
        <purchaseOrderDet>
            <line>02</line>
            <dueDate>2002-06-01</dueDate>
            <part>10-15000</part>
            <qtyOrd>20</qtyOrd>
            ...
        </purchaseOrderDet>
    </purchaseOrder
    <qcom:exception>
        <qcom:number>864</qcom:number>
        <qcom:description>SITE ADDRESS DOES NOT EXIST
        </qcom:description>
        <qcom:severity>error</qcom:severity>
        <qcom:field>site</qcom:field>
        <qcom:context>PO1234</qcom:context>
    </qcom:exception>
</maintainPurchaseOrderResponse>
```

The following XML schema fragment describes the syntax of the previous sample. The `PurchaseOrderResponseType` definition is used in the QDoc response and is stored in a separate file and included in the top-level schema. This defaults to include primary fields and can be replaced by a user-defined file that contains user-defined fields.

```
    <complexType name="PurchaseOrderResponseType">
        <sequence>
            <element name="nbr" type="string" minOccurs="0">
```

```
            <annotation>
                <documentation>Purchase Order</documentation>
            </annotation>
        </element>
        <element name="vend" type="string" minOccurs="0">
            <annotation>
                <documentation>Vendor</documentation>
            </annotation>
        </element>
        <element name="ordDate" type="string" minOccurs="0">
            <annotation>
                <documentation>Order Date</documentation>
            </annotation>
        </element>
        <element name="rmks" type="string" minOccurs="0">
            <annotation>
                <documentation>Remarks</documentation>
            </annotation>
        </element>
        <element name="purchaseOrderCmt" type="qdoc:PurchaseOrderCmtResponseType"
        minOccurs="0" maxOccurs="unbounded"/>
        <element name="purchaseOrderDet" type="qdoc:PurchaseOrderDetResponseType"
        minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
</complexType>
<complexType name="PurchaseOrderCmtResponseType">
    <sequence>
        <element name="seq" type="string" minOccurs="0">
            <annotation>
                <documentation>Sequence</documentation>
            </annotation>
        </element>
        <element name="cmmt" type="string" minOccurs="0">
            <annotation>
                <documentation>Trans Comment</documentation>
            </annotation>
        </element>
    </sequence>
</complexType>
<complexType name="PurchaseOrderDetResponseType">
    <sequence>
        <element name="line" type="integer" minOccurs="0">
            <annotation>
                <documentation> Ln</documentation>
            </annotation>
        </element>
        <element name="dueDate" type="integer" minOccurs="0">
            <annotation>
                <documentation> Due Date</documentation>
            </annotation>
        </element>
        <element name="part" type="integer" minOccurs="0">
            <annotation>
                <documentation> Part</documentation>
            </annotation>
        </element>
        <element name="qtyOrd" type="integer" minOccurs="0">
            <annotation>
                <documentation> Quantity Ordered</documentation>
            </annotation>
        </element>
    </sequence>
</complexType>
```

QDoc Message Envelope

Q/LinQ currently stores all control information regarding its export and import documents in non-XML data known as document control tags. While it is able to both export and import documents preceded by these tags, the tag format is non-XML and Q/LinQ-specific. These tags are not processed easily by XML-based tools such as XSLT stylesheets.

A standard, QAD-specific message envelope for use by QXtend as well as Q/LinQ is defined in this section using SOAP 1.0 syntax. It includes a minimal set of elements that must be understood by both Q/LinQ and QXtend, although additional product-specific elements can be added to it as needed in order to support interoperability with third parties. Its content is also loosely based on the envelopes/headers prescribed by the JMS, OAGIS, and ebXML specifications.

The present version of the QDoc envelope is likely to change significantly in future releases, as future XML-based messaging standards are published and adopted over time by the software industry.

While the QDoc message envelope is designed primarily to contain QDocs, it can also be used to contain any other type of document exported or imported by Q/LinQ such as OAGIS BODs. Its use is required for all QDocs.

Envelope Syntax

Based on SOAP standards, all QDoc messages consist of the XML root element `Envelope` with two child elements, `Header` and `Body`. The `Body` element contains a single QDoc as described in the preceding sections as its only payload. The `Header` element consists of one or more header block elements named `qdoc` as described in the following section.

The `Envelope`, `Header`, and `Body` elements are defined and must be labeled using the following SOAP Envelope namespace URI:

```
http://schemas.xmlsoap.org/soap/envelope/
```

They may contain attributes, but these may be ignored.

Header Block Content

This section describes the header block content of QDocs. The information in this section applies to both QDoc specification versions, except where indicated.

Each QDoc header block consists of a root element QDoc containing the elements described in this section. Header blocks may also contain several other attributes:

- In SOAP 1.0 the `role` attribute may have any legal URI as its value. In SOAP 1.1 this attribute is replaced with the `actor` attribute.
- In SOAP 1.0 the `mustUnderstand` attribute is Boolean. In SOAP 1.1 this attribute has a value of either 1 (true) or 0 (false). The absence of the SOAP `mustUnderstand` attribute is semantically equivalent to its presence with the value “0”.

  Both of these attributes are ignored on inbound QDocs.
- The `encodingStyle` attribute is a URI, and is set to the following value on all outbound QDoc header blocks:

  http://www.w3.org/2002/12/26/soap-encoding

This attribute is not used in SOAP 1.1 QDocs.

While more than one header block inside a single message is syntactically allowed, in the case of inbound QDocs all QDoc headers after the first one are ignored. All outbound QDocs contain only a single QDoc header block.

Requestor Element

Note The `requestor` element in SOAP 1.1 QDocs has been moved from the header block to `dsSessionContext` along with requestor password. For details see “Session Context” on page 268.

The `requestor` element can contain user ID and password attributes. If authentication credentials are provided in the QDoc, QXI will always use these credentials to verify the QDoc regardless of the setting of `useQDocRequestor`. This helps in situations where the application needs a detailed audit trail by user, but you do not want to require all QDocs to have authentication information in them.

You should note the following:

- If `useQdocRequestor` is set to false and user name and password nodes are specified in the QDoc, QXI will use them for authentication. If the nodes are not specified, QXI will use the user name/password combination in the connection pool setting for authentication. If you do not want Inbound to use the user name and password in QDoc, you have to remove the nodes from QDoc.
- If `useQdocRequestor` is set to true, QXI always uses the user name and password in the request QDoc.

Note In previous versions, QXI used the credentials in the QDoc only if `useQDocRequestor` was enabled; otherwise the credentials specified in the connection pool were used.

For XML syntax 1.1 QDocs, if the username and password should not be used in the request, the following nodes must be removed from QDoc:

```
<qcom:ttContext>
    <qcom:propertyQualifier>QAD</qcom:propertyQualifier>
    <qcom:propertyName>username</qcom:propertyName>
    <qcom:propertyValue/>
</qcom:ttContext>
<qcom:ttContext>
    <qcom:propertyQualifier>QAD</qcom:propertyQualifier>
    <qcom:propertyName>password</qcom:propertyName>
    <qcom:propertyValue/>
</qcom:ttContext>
```

For XML syntax 1.0 Qdocs, if the username and password should not be used in the request, the following nodes must be removed from QDoc:

```
<requestor/>
```

If the current request QDoc sent from a product has empty user name and password, these must be removed from the QDoc. If they are not removed, QXI will use them for authentication and return the error `Invalid Username and Password`.

This password is sent in text form, encrypted within the server, and is encrypted in any response QDocs.

The `requestor` element can be omitted from the header if the QXtend implementation does not require user ID and password.

Note At the receiver level you can specify that every QDoc that is for a receiver must have authentication information in it. For details see “Adding Inbound Receivers” on page 181.

sessionId

Note The information in this section applies only to the 1.1 specification and QAD EE.

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.

senderId

Note In the QDoc 1.1 specification Web Services Addressing (WS-Addressing) is used and consequently there is no `senderId` (or `receiverId`). The information in this section applies only to the 1.0 specification.

The `senderId` element identifies the sender of the QDoc or other document and is defined as a URI.

While most of the URI is not yet used, it must include the query parameter `sender` in order to identify the sending entity within a given installation. The syntax of the query parameters is similar to that commonly used with any URI, as in the following example:

```
http://anyBaseURI/path1/path2?sender=myCompany&amp;…
```

The ampersand character separating the query parameters must be written as `&` in order to pass XML well-formedness parsing rules.

In the case of outbound documents sent by Q/LinQ, sender is always equal to the Q/LinQ System ID (@SYSID tag) maintained in the Q/LinQ System Control table. This tag must have the same value as the configuration parameter used by the QDoc server to identify a given QAD Enterprise Applications database in a multi-database installation.

In the case of inbound documents, sender is stored in the Q/LinQ trading partner ID (@TRADPTRID tag) field. The trading partner ID identifies the external owner of the QAD Enterprise Applications object about which Q/LinQ is being notified.

receiverId

Note The `receiverId` element has been removed in the 1.1 QDoc specification. In QDoc specification 1.0 the receiver was passed as a parameter in the URL in the receiverId node. In QDoc specification 1.0 the receiver is now the third entry in the URN of the `To` node:

```
<ns0:To xmlns:ns0="http://www.w3.org/2005/08/addressing">urn:services-qad-
com:QADSE</ns0:To>
```

receiverId in QDoc Specification 1.0

The `receiverId` element identifies the receiver of the QDoc or other document and is defined as a URI.

While most of the URI is not yet used, it must include the query parameters `connection` and `receiver` in order to identify the receiving entity within a given installation. The syntax of the query parameters is similar to that commonly used with any URI, as in the following example:

```
http://anyBaseURI/path1/path2?connection=yourQueue&amp;receiver=
yourCompany&amp;…
```

In addition, the URI can also include the optional query parameter `domain` in order to identify the receiving domain within the QAD Enterprise Applications database.

Note In the QDoc 1.1 specification the parameter `domain` has been moved from `receiverId` to `dsSessionContext` as a propertyName of `domain`. For details see “Session Context” on page 268.

The syntax of this query parameter is shown in the following example:

```
http://anyBaseURI/path1/path2?connection=yourQueue&amp;receiver=
yourCompany&amp;domain=yourdomain&amp;
```

If this optional parameter is omitted, the QDoc will use the domain that is specified in the Domain field in the Connection Pool Manager for that procedure. See “Configuring Connection Pools” on page 204.

In the case of outbound documents sent by Q/LinQ, the receiver is the Q/LinQ trading partner ID (@TRADPTRID tag) for which the export document was published. In many cases, this value corresponds to the external owner of the QAD Enterprise Applications data object that is the main subject of the document. The connection is the Q/LinQ application ID (@APPID tag) through which the document was sent.

In the case of Q/LinQ inbound documents, the receiver must be set to the Q/LinQ System ID (@SYSID tag) field maintained in the Q/LinQ System Control table, which in turn must be the same as the identifier used in the QDoc server to identify the target QAD Enterprise Applications database. However, it is ignored by the current version of Q/LinQ, as each Q/LinQ installation is associated with one and only one QAD Enterprise Applications database and cannot properly route documents to any other QAD Enterprise Applications database. The connection is defined by the sender, but is mapped by Q/LinQ to the Q/LinQ application ID (@APPID tag) through which the document is received.

senderDocumentId

The `senderDocumentId` element is the sender’s locally unique identifier for the QDoc, meaningful to the sender’s messaging software but not necessarily to the source application. It is required and must be referenced in the `originalDocumentRef` field on QDoc confirmations so that the receiver of the confirmation—the sender of the original document—can unambiguously identify the document being confirmed.

The `senderDocumentId` element is the Q/LinQ internally assigned document ID. It is not significant outside of Q/LinQ, but it provides a potentially useful cross-reference back to the Q/LinQ database.

Note This element has been removed in the 1.1 QDoc specification. In this specification each message uses `MessageID` to convey the `MessageID` property:

```
<ns0:MessageID xmlns:ns0="http://www.w3.org/2005/08/addressing">urn:messages-qad-
com::2007-02-08T14:10:11.424-08:00:QADSE:</ns0:MessageID>
```

documentStandard

The `documentStandard` element is intended to identify the standard or specification, if any, to which the document conforms. This could be an externally published standard such as EDIFACT, OAGIS, ANSI X12, or UCCnet. It could also be an internal identifier for a proprietary grammar used at one particular installation, for those QAD application users who have defined standard business objects for use inside their enterprise.

The special value QDoc is reserved by QAD to designate QDocs. `documentStandard` is an optional element.

This element is exactly equivalent to the Q/LinQ document standard or @DOCSTD document control tag. As with Q/LinQ, the special value `cim` is reserved by QAD to designate documents that can be processed by the QAD Enterprise Applications CIM Interface or the UI-emulation mode of Q/LinQ.

Note In the QDoc 1.1 specification the `documentStandard` element has been moved into the SOAP header `ReferenceParameters` section.

documentType

The `documentType` element identifies the type of document contained in the message payload (that is, the SOAP `Body`), and is used to route inbound documents to the appropriate handler for processing. It is semantically equivalent to a message type, action, or service designator. It is an optional element, and may be assigned a default value by the QAD Integration Framework based on the needs of the installation.

For QDocs, this element should be set to the name of the QDoc; for example, `maintainSalesOrder`.

This element is exactly equivalent to the Q/LinQ document type or @DOCTYP document control tag. For documents with a `documentStandard` of `cim`, it should be set to the name of the QAD Enterprise Applications program called to process the document.

Note In the QDoc 1.1 specification the `documentType` element has been moved into the SOAP header `ReferenceParameters` section.

documentVersion

The `documentVersion` element qualifies the `documentType` with a version or revision designator that permits different variations of the same type of document to be distinguished for special handling. For documents conforming to an externally published standard, such as EDI, it is set to the version or revision level designator assigned by the published standard.

If `documentVersion` is omitted, it is presumed to designate the most current commercial or public version of the associated document type. When used with internal, less formal document grammars that do not use any form of version control, it would be omitted.

For QDocs, this element should be set to the version attribute of the QDoc; for example, eB_1.

This element is exactly equivalent to the Q/LinQ document revision or @DOCREV document control tag.

Note In the QDoc 1.1 specification the `documentVersion` element has been moved into the SOAP header `ReferenceParameters` section.

confirmationLevel

The `confirmationLevel` element designates under what circumstances a QDoc Confirmation document is requested. Allowed values are none, error, or all.

This element has exactly the same meaning as the present Q/LinQ tag @ACKLVLREQD.

Note The `confirmationLevel` element has been removed in the QDoc 1.1 specification.

dateTimeCreated

The `dateTimeCreated` element contains the date- and time-stamp for the creation of the document, expressed using the `DateTime` type prescribed by XML schema. For example, the following value would apply to a document created on July 1, 2002, at 9:00am Pacific Standard Time:

2002-07-01T09:00:00-0800

This element has exactly the same meaning as the present Q/LinQ tags @DATECREATE, @TIMECREATE, and @TIMEZONE in combination.

Note The `dateTimeCreated` element has been removed in the QDoc 1.1 specification.

senderDocumentRef

The `senderDocumentRef` is a flexible-format, multi-occurrence string providing a reference back to the sending application. It is not necessarily unique and serves only to provide a meaningful reference for application end users, as opposed to the `senderDocumentId` used only for document matching purposes. Accordingly, it should be assigned to one or more values that represent the key fields of the application data objects most closely associated with the document.

For example, in the case of a sales order document it might contain the sales order number. In the case of inventory balances, it might contain the SKU number of the material as well as the identifier of the warehouse/facility in which the material is stored.

Q/LinQ populates only two occurrences of this field per document at most. If the corresponding QAD Enterprise Applications data object has a primary site code associated with it, as in the case of inventory, the first occurrence is set to the QAD Enterprise Applications site (@MFGPROSITE) tag and the second occurrence to the QAD Enterprise Applications key (@MFGPROKEY) tag. Otherwise, the first occurrence is set to the QAD Enterprise Applications key (@MFGPROKEY) tag.

Note The `senderDocumentRef` element has been removed in the QDoc 1.1 specification.

receiverDocumentRef

The `receiverDocumentRef` is a flexible-format, multi-occurrence string providing a reference back to the receiving application. It is not necessarily unique and serves only to provide a meaningful reference for application end users. It is analogous to the `senderDocumentRef` field, and should be assigned to one or more values that represent the key fields of the application data objects most closely associated with the document.

In most cases, this field is omitted from the message envelope, as the sender application or messaging agent would not normally have a meaningful document reference expressed in terms of the receiver. However, it may be useful for follow-up messages in which two applications are engaged in a long-running dialog about data objects known to both. For example, QAD Enterprise Applications might use this field on a shipment notification to reference an external sales order number and line.

Q/LinQ expects only two occurrences of this field per document at most:

- If the corresponding QAD Enterprise Applications data object has a primary site code associated with it, as in the case of inventory, the first occurrence is assumed to be a site code and is stored in the QAD Enterprise Applications site (@MFGPROSITE) tag.
- The second occurrence is assumed to be some other key field and is stored in the QAD Enterprise Applications key (@MFGPROKEY) tag. Otherwise, the first occurrence is stored in the QAD Enterprise Applications key (@MFGPROKEY) tag.

Note The `receiverDocumentRef` element has been removed in the QDoc 1.1 specification.

SOAP Compliance Limitations

The QDoc message envelope has the following limitations and qualifications:

- The QDoc solution supports document-style message exchange as opposed to SOAP RPC. In order to make support for SOAP RPC and related standards such as WSDL easier to implement in future releases, SOAP encoding rules are used to represent all QDoc data.

  Note The above limitation applies to the QDoc 1.0 specification only. The QDoc 1.1 specification does not use SOAP encoding rules.
- All QDoc messages require a `Header` element, whereas SOAP considers this element optional.
- QAD applications ignore the `role` attribute on all SOAP 1.0 header blocks—or the `actor` attribute in SOAP 1.1 headers—acting as the ultimate SOAP receiver in all cases. In other words, a QAD application automatically plays all roles that appear in the SOAP message.
- QAD applications ignore the `mustUnderstand` attribute on all SOAP header blocks, assuming that it must understand and process every received QDoc. In other words, they always act as if `mustUnderstand` is set to true (for SOAP 1.0) or 1 (for SOAP 1.1).
- QAD applications cannot act as SOAP intermediaries, passing the QDoc to other SOAP nodes for processing. Rather, they process all inbound messages as the ultimate receiver.
- QAD applications process a received QDoc only once, regardless of the number of SOAP header blocks that may be included inside the `Header` element of the envelope.

Future Extensions and Forward Compatibility

For QDocs using the 1.0 standard, the content of the QDoc envelope—while mostly SOAP compliant—is entirely QAD defined with no reference to external industry standards or specifications outside of SOAP. Qdocs using the 1.1 standard follow the Web Service Addressing (WSA) standard. As XML- and SOAP-based messaging standards mature, the QDoc envelope will evolve to address authentication, privacy, reliability, and work flow requirements among others.

QDoc Envelope Examples

The following examples illustrates QDoc envelopes for both the 1.1 and 1.0 syntax specifications.

Envelope Example (1.1)

```
<!-- Here is an inbound QDoc 1.1 request message envelope. -->
<?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:eb21sp6</wsa:To>
        <wsa:MessageID>urn:services-qad-com::eb21sp6</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>
        <!-- The QDoc goes here -->
        <maintainSalesOrder>
            ...
        </maintainSalesOrder>
    </soapenv:Body>
</soapenv:Envelope>
```

Envelope Example (1.0)

```
<!-- Here is an inbound QDoc 1.0 request message envelope. -->
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance">
    <env:Header>
        <!-- SOAP 'mustUnderstand' and 'role' attributes are ignored but may be present -->
        <QDoc xmlns="http://www.qad.com/qdoc/common" env:mustUnderstand="0" env:actor=
        "http://schemas.xmlsoap.org/soap/actor/next">
            <senderId>http://www.acme.com/site1?sender=site1</senderId>
            <receiverId>http://www.acme.com/site2?connection=queue1&amp;receiver=
            db2</receiverId>
            <senderDocumentId>eq200000</senderDocumentId>
            <requestor id="user" password="apasswd" type="mfgpro"/>
            <!-- descriptor may be omitted if Q/LinQ is set up to expect QDocs -->
            <documentStandard>QDoc</documentStandard>
            <documentType> maintainSalesOrder</documentType>
            <documentVersion/>
            <confirmationLevel>all</confirmationLevel>
            <dateTimeCreated>2002-07-10T09:00:00-0800</dateTimeCreated>
            <senderDocumentRef>eqso5678</senderDocumentRef>
            <senderDocumentRef>1</senderDocumentRef>
        </QDoc>
    </env:Header>
    <env:Body>
        <!-- The QDoc goes here -->
        <maintainSalesOrder …>
                ...
```

```
        </maintainSalesOrder>
    </env:Body>
</env:Envelope>
```

XML Schema for Envelope Header

Following is the unannotated XML 1.0 schema describing the syntax of the header block of the QDoc document envelope. It makes no distinction between export and import, so that exported Q/LinQ documents can be reprocessed as import documents with no XML syntax errors. The elements and attributes are defined in an envelope-specific namespace rather than the primary product-specific QDoc namespaces.

Note In the QDoc 1.1 specification there is no common schema; instead, each QDoc has its own schema. For details of changes to the XML schema in the 1.1 specification see “QDoc Schema 1.1” on page 267.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.qad.com/qdoc/common" xmlns:qcom=
"http://www.qad.com/qdoc/common" xmlns="http://www.w3.org/2001/
XMLSchema" xmlns:soapenv="http://www.w3.org/2002/12/soap-envelope" xmlns:enc=
"http://www.w3.org/2002/12/soap-encoding" elementFormDefault="qualified" xml:lang="EN">
    <import namespace="http://www.w3.org/2002/12/soap-envelope" SchemaLocation="soap-
    envelope.xsd"/>
    <complexType name="QdocEnvelopeType">
        <sequence>
            <element name="senderId" type="anyURI"/>
            <element name="senderRef" type="string" minOccurs="0"/>
            <element name="receiverId" type="anyURI"/>
            <element name="receiverRef" type="string" minOccurs="0"/>
            <element name="senderDocumentId" type="string"/>
            <element name="documentStandard" type="string"/>
            <element name="documentType" type="string"/>
            <element name="documentVersion" type="string"/>
            <element name="confirmationLevel" default="none" minOccurs="0">
                <simpleType>
                    <restriction base="string">
                        <enumeration value="none"/>
                        <enumeration value="error"/>
                        <enumeration value="all"/>
                    </restriction>
                </simpleType>
            </element>
            <element name="dateTimeCreated" type="dateTime" minOccurs="0"/>
            <element name="senderDocumentRef" type="string" minOccurs="0" maxOccurs=
            "unbounded"/>
            <element name="receiverDocumentRef" type="string" minOccurs="0" maxOccurs=
            "unbounded"/>
            <element name="requestor" minOccurs="0" maxOccurs="unbounded">
                <complexType>
                    <attributeGroup ref="qcom:requestorAttributes"/>
                </complexType>
            </element>
            <any minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
        <attribute ref="soapenv:encodingStyle"/>
        <attribute ref="soapenv:role"/>
        <attribute ref="soapenv:mustUnderstand"/>
        <anyAttribute/>
    </complexType>
    <element name="qdoc" type="qcom:QdocEnvelopeType"/>
    <attributeGroup name="requestorAttributes">
        <attribute name="id" type="string" use="optional"/>
        <attribute name="type" type="string" use="optional"/>
        <attribute name="password" type="string" use="optional"/>
    </attributeGroup>
    <attributeGroup name="commonAttributes">
        <attribute name="version" type="string" use="required"/>
```

```
    <attribute name="scopeTransaction" type="boolean" use="optional"/>
    <attribute name="logTransaction" type="boolean" use="optional"/>
    <attribute name="transactionId" type="string" use="optional"/>
    <attribute name="suppressResponseDetail" type="boolean" use="optional"/>
    <attribute ref="xml:lang"/>
</attributeGroup>
<attributeGroup name="arrayEntryAttributes">
    <attribute name="index" type="positiveInteger" use="required"/>
    <attribute name="skip" type="boolean"/>
</attributeGroup>
<complexType name="EntryString">
    <simpleContent>
        <extension base="string">
            <attributeGroup ref="qcom:arrayEntryAttributes"/>
        </extension>
    </simpleContent>
</complexType>
<complexType name="EntryInteger">
    <simpleContent>
        <extension base="integer">
            <attributeGroup ref="qcom:arrayEntryAttributes"/>
        </extension>
    </simpleContent>
</complexType>
<complexType name="EntryDecimal">
    <simpleContent>
        <extension base="decimal">
            <attributeGroup ref="qcom:arrayEntryAttributes"/>
        </extension>
    </simpleContent>
</complexType>
<complexType name="EntryDate">
    <simpleContent>
        <extension base="date">
            <attributeGroup ref="qcom:arrayEntryAttributes"/>
        </extension>
    </simpleContent>
</complexType>
<complexType name="ExceptionType">
    <sequence>
        <element name="number" type="string"/>
        <element name="description" type="string"/>
        <element name="severity" default="error" minOccurs="0">
            <simpleType>
                <restriction base="string">
                    <enumeration value="informational"/>
                    <enumeration value="warning"/>
                    <enumeration value="error"/>
                </restriction>
            </simpleType>
        </element>
        <element name="field" type="string" minOccurs="0"/>
        <element name="context" type="string" minOccurs="0"/>
        <element name="trace" type="string" minOccurs="0"/>
    </sequence>
</complexType>
<simpleType name="ReturnValueType">
    <restriction base="string">
        <enumeration value="success"/>
        <enumeration value="warning"/>
        <enumeration value="error"/>
    </restriction>
</simpleType>
<complexType name="QdocConfirmationType">
    <sequence>
        <element name="originalDocumentRef" type="qcom:QdocEnvelopeType"/>
        <element name="processingStage">
            <simpleType>
                <restriction base="string">
                    <enumeration value="received"/>
                    <enumeration value="accepted"/>
                    <enumeration value="processed"/>
```

```
                </restriction>
            </simpleType>
        </element>
        <element name="processingOutcome">
            <simpleType>
                <restriction base="string">
                    <enumeration value="error"/>
                    <enumeration value="warning"/>
                    <enumeration value="success"/>
                </restriction>
            </simpleType>
        </element>
        <element name="exception" type="qcom:ExceptionType" minOccurs="0" maxOccurs=
        "unbounded"/>
        <element name="processingResults" minOccurs="0">
            <complexType>
                <sequence>
                    <any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
                </sequence>
            </complexType>
        </element>
    </sequence>
    <attributeGroup ref="qcom:commonAttributes"/>
</complexType>
</schema>
```

SOAP Faults

As previously discussed, any errors detected in the preprocessing of an inbound QDoc are thrown to the sender as SOAP faults. SOAP faults are SOAP response messages that contain a single instance of the `Fault` element inside the SOAP body, with no other data content.

The purpose of SOAP faults is to describe a single fatal error to the sender of the original SOAP message using a standard syntax.

Note Aside from the commonality mentioned above, considerable differences exist in the way that SOAP faults are handled in SOAP 1.0 and SOAP 1.1. QXtend version 1.4 uses SOAP 1.1 syntax; QXtend version 1.3 and below use SOAP 1.0 syntax. Refer to the appropriate section for the version of QXtend being used.

SOAP Faults in SOAP 1.1

The SOAP `Fault` element defines the following four subelements:

- `faultcode` (mandatory). The `faultcode` element is intended to allow software to provide a way to identify the fault. The `faultcode` value must be a qualified name; see page 313 for descriptions of `faultcode` names. SOAP defines the following SOAP fault codes covering basic SOAP faults:
  - `VersionMismatch`. For details see the section “SOAP Faults in SOAP 1.0” on page 313.
  - `MustUnderstand`. For details see the section “SOAP Faults in SOAP 1.0” on page 313.
  - `Client`. Indicates that the message was not well-formed or that it did not contain the appropriate information in order to succeed. This fault code indicates that the message requires modification before being resent.
  - `Server`. Indicates that the message was not processed because of errors—communication problems with servers, for example—not related to the contents of the message itself. The message might succeed if sent later.

- `faultstring` (mandatory). The `faultstring` element is intended to provide a human-readable explanation of the nature of the fault.
- `faultactor`. The `faultactor` element is intended to identify who caused the fault within the message path. It is similar to the SOAP `actor` attribute but instead indicates the source of the fault. The value of the `faultactor` attribute is a URI identifying the source. The `faultactor` element is not used as it is presumed that the QAD application is always the ultimate receiver of the SOAP message.
- `detail` (mandatory). The detail element is intended to carry application-specific error information related to the `Body` element. If the `detail` element is absent, this indicates that the fault is not related to processing of the `Body` element. This can be used to distinguish whether the `Body` element was processed or not in case of a fault situation.

  Immediate child elements of the `detail` element are called detail entries. Each detail entry is encoded as an independent element within the detail element.

Other `Fault` subelements may be present provided they are namespace-qualified.

This partial example of an outbound QAD Enterprise Applications QDoc response contains a SOAP fault.

```
<!-- Here is an outbound QDoc response message envelope
containing a SOAP fault. -->
<?xml version="1.0" ?>
<soapenv:Envelope xmlns:soapenv= "http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsa=
"http://www.w3.org/2005/08/addressing" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header>
        ...
    </soapenv:Header>
    <soapenv:Body>
        <!-- The SOAP fault follows, with no other QDoc content
            allowed -->
        <soapenv:Fault xmlns="http://schemas.xmlsoap.org/soap/envelope/">
            <soapenv:faultcode>Client.qdocVersionNotSupported
            </soapenv:faultcode>
              <soapenv:faultstring>QDoc is not supported</soapenv:faultstring>
              <soapenv:faultactor />
            <soapenv:detail>
                <ns1:dsExceptions xmlns:ns1="urn:schemas-qad-com:xml-services:common">
                    <ns2:temp_err_msg xmlns:ns2="urn:schemas-qad-com:xml-services">
                        <ns1:tt_msg_nbr>QdocExceptionex005</ns1:tt_msg_nbr>
                          <ns1:tt_msg_desc>QDoc is not supported</ns1:tt_msg_desc>
                          <ns1:tt_msg_sev>error</ns1:tt_msg_sev>
                         <!-- field and context are not relevant to this type of error -->
                          <ns1:tt_msg_field />
                          <ns1:tt_msg_context />
                          <ns1:tt_msg_data>QdocException: Qdoc Receiver is not recognized.
                        The Receiver is not setup in the Qdoc Server instance at
                        com.qad.qxtend.qdocs...
                            ...
                        </ns1:tt_msg_data>
                      <ns1:tt_msg_keys />
                      <ns1:tt_msg_keys />
                      <ns1:tt_msg_keys />
                      <ns1:tt_msg_keys />
                      <ns1:tt_msg_keys />
                      <ns1:tt_msg_keys />
                      <ns1:tt_msg_datetime>2007-05-31T17:54:18+1000</ns1:tt_msg_datetime>
                    <ns1:tt_msg_processed>false</ns1:tt_msg_processed>
                      <ns1:tt_level />
                      <ns1:tt_msg_index>0</ns1:tt_msg_index>
                  </ns2:temp_err_msg>
                </ns1:dsExceptions>
            </soapenv:detail>
```

```
        </soapenv:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

SOAP Faults in SOAP 1.0

The `Fault` element contains:

- Mandatory `Code` element
- Mandatory `Reason` element
- Optional `Node`, `Role`, and `Detail` elements

The `Code` element values for common errors, as well as related `Subcode` element values, are predefined in the SOAP 1.0 specification:

- `VersionMismatch`: The message envelope does not conform to the SOAP specification.
- `MustUnderstand`: A `Header` element was not understood even though the SOAP `mustUnderstand` attribute was set to true. This is not applicable to QDocs, as the `mustUnderstand` attribute is not used.
- `DataEncodingUnknown`: The SOAP `encodingStyle` attribute has a value other than:

  http://www.w3.org/2002/12/soap-encoding
- `Sender`: The QDoc was incorrectly formed by the sender in some fashion not covered by the other general error conditions, as further described by the `Subcode` element.
- `Receiver`: The QDoc could not be processed by the receiver for reasons other than its syntax or content, as further described by the `Subcode` element.

The following `Subcode` value elements are used with the `Sender` code to describe errors in the QDoc request syntax:

- `QDocNotWellFormed`: The QDoc body's XML content is not well formed and cannot be parsed.
- `QDocEnvelopeInvalid`: The QDoc's envelope does not have valid syntax, and the contained QDoc cannot be unwrapped.
- `QDocNotRecognized`: No QDoc with the given name is supported by the installation.
- `QDocSyntaxInvalid`: The QDoc is recognized and its XML content is well formed, but it does not have valid syntax with respect to its XML schema. For example, the wrong input parameters are provided.
- `QDocReceiverNotRecognized`: The `receiver` attribute of the QDoc message envelope was not recognized; therefore, the QDoc cannot be processed.
- `QDocVersionNotSupported`: No QDoc is supported by the installation with the given combination of `name`, `version`, and `receiver` attributes.

The following `Subcode` value element is used with the `Receiver` code to describe errors in the QAD application responsible for processing the QDoc request:

- `QDocApplicationNotAvailable`: The QAD application responsible for processing the QDoc request is not available. For example, the QAD Enterprise Applications database server is not running, or the QDoc server is not able to communicate with Progress clients or AppServers.

The `Detail` element of a QDoc SOAP `Fault` contains one or more `exception` elements of the type `ExceptionType` describing the nature of the preprocessing problem. In all such exceptions, the number element is blank, severity is error, and description contains an error message describing the problem. The possible error messages consist of text equivalent to the QDoc-specific `Subcode` values listed here.

The SOAP `Reason` element is set to the value of the description element of exception, essentially the text of the error message that would be displayed to a human user. The SOAP `Node` and `Role` elements are not needed or used, as it is presumed that the QAD application is always the ultimate receiver of the SOAP message.

The following XML schema fragment defines the QAD-specific `Subcodes`.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
           xmlns:qcom="http://www.qad.com/qdoc/common"
           targetNamespace="http://www.qad.com/qdoc/common">
    <element name="soapFaultSubcodeValue">
        <simpleType>
            <restriction base="QName">
               <enumeration value="qcom:qdocNotWellFormed"/>
               <enumeration value="qcom:qdocEnvelopeInvalid"/>
               <enumeration value="qcom:qdocNotRecognized"/>
               <enumeration value="qcom:qdocSyntaxInvalid"/>
               <enumeration value="qcom:qdocReceiverNotRecognized"/>
               <enumeration value="qcom:qdocVersionNotSupported"/>
               <enumeration
                   value="qcom:qdocApplicationNotAvailable"/>
            </restriction>
        </simpleType>
    </element>
</schema>
```

This partial example of an outbound QAD Enterprise Applications QDoc response contains a SOAP fault.

```
<!-- Here is an outbound QDoc response message envelope
    containing a SOAP fault. -->
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
    xmlns:env="http://www.w3.org/2002/12/soap-envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <env:Header>
        ...
    </env:Header>
    <env:Body>
        <!-- The SOAP fault follows, with no other QDoc content
            allowed -->
        <env:Fault>
            <env:Code>
               <env:Value>env:Sender</env:Value>
               <env:Subcode>
                   <!-- Subcode value is QAD-defined -->
                   <env:Value
xmlns:qcom="http://www.qad.com/qdoc/common">
                       QDocNotWellFormed
                   </env:Value>
               </env:Subcode>
            </env:Code>
            <!-- Reason is taken from the QDoc exception description
               -->
            <env:Reason xml:lang="en-US">
               QDoc is not well formed
            </env:Reason>
            <env:Detail>
               <!-- Here is the QDoc exception element -->
               <qcom:exception
                   xmlns:qcom="http://www.qad.com/qdoc/common">
```

```
                    <qcom:number></qcom:number>
                    <qcom:description>
                        QDoc is not well formed
                    </qcom:description>
                    <qcom:severity>error</qcom:severity>
                    <!-- field and context are not relevant to this
                        type of error -->
                    <qcom:field/>
                <qcom:context/>
            </qcom:exception>
            </env:Detail>
            ...
        </env:Fault>
    </env:Body>
</env:Envelope>
```

Chapter 25

Telnet Reference

This section provides information on telnet configuration and security.

Introduction

QAD QXtend Inbound (QXI) connects to the QAD Enterprise Applications receivers using telnet. This connection is managed in the Connection Pool Manager. If QXI and the QAD Enterprise Applications receivers are installed on the same machine, the Connection Pool Manager still uses telnet to connect the QAD QXtend instance to the QAD Enterprise Applications instance.

During normal operation, the sequence of events is as follows:

1. Start the database servers for the QAD Enterprise Applications instances.
2. Start QXI.
3. QXI Connection Pool Manager creates connections to QAD Enterprise Applications. This connection is broken down into:
   a. Log in to the database server.
   b. Move to the QAD Enterprise Applications launch directory.
   c. Launch QAD Enterprise Applications using special run scripts.

This section provides the necessary details on entering the correct telnet log-in script into the Connection Pool Manager, and how to create the special QAD Enterprise Applications run scripts.

Additional information is available at the end of the section on telnet security.

Creating Telnet Log-In Scripts

The value entered in the Server Startup Script field in Configuration Update Settings in the Connection Pool Manager must be the exact log-in sequence for the telnet session including both system prompts and responses. See “Adding a Connection Pool” on page 204.

The objective of the script is to log in to a telnet session on the database server machine and to connect to a specific running QAD Enterprise Applications database instance. Use the following sample script and descriptions to create the correct script for your environment.

Prompts and responses are separated using the pipe symbol (|). The prompt values are case sensitive. The values you enter must be identical to prompts the telnet server displays when the Connection Pool Manager attempts to log in. Verify each step manually before attempting to start Connection Pool Manager.

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

`login:` is the first prompt displayed by the system.

`QXtend` is the response to this prompt, the user ID used to log in to the system.

`Password`: is the second prompt displayed.

`$PASSWD`: is the alias for the server startup password. This is used for added security measures. If you do not want the password to be displayed in clear text, then enter the password into the Server Startup Password field in the Connection Pool configuration parameters screen and this alias will be replaced by your password once it is decrypted. Hard coding the actual password in place of this alias will also function; however the password will be clearly visible to all. For password entry instructions, see “Adding a UI API Pool” on page 205.

`$` is the prompt displayed after successful log-in, indicating the system is ready.

`exec ./qma.QXprod` is the command to execute the connection script on a UNIX system. The `exec` is required to ensure that the script does not open in a new window.

`qma.QXprod` script is a copy of the standard `production` script, with the `mfgwrapper` and `apimode` parameter added to the appropriate command line. See the next section for details.

On Windows systems, an equivalent log-in script would look like:

```
login:|QXtend|password:|$PASSWD|c:\mfgsvr\telnet>| startmfg.bat
```

Editing QAD Enterprise Applications Telnet Scripts

The telnet log-in script entered in the Connection Pool Manager calls the telnet connection script for a specific instance of QAD Enterprise Applications. See the “Generating Scripts” heading in the Character Client Install section of your QAD Enterprise Applications installation guide. The telnet connection scripts are first generated by MFG/UTIL during the installation of QAD Enterprise Applications as `qma.DBSetName` scripts, where *`DBSetName`* is the name of the database to which the script connects. For example, the telnet connection script for the Production database set is `qma.Production`.

In order for the QXtend Connection Pool Manager to connect to an QAD Enterprise Applications instance using the `qma` script for a database, modifications to the script are required.

The telnet connection scripts are located in your QAD Enterprise Applications database server administration directory.

In general, the connection script performs the following functions:

- Changes to the user’s home directory
- Sets environment requirements and variables
- Provides connection parameters for the databases in the set

Note Template scripts are deployed with QAD QXtend and can be found under the `adapter` directory.

Sample Connection Manager Telnet Script

The following is a UNIX `qma` script. The bold sections are discussed in greater detail below.

```
#!/bin/sh
# Script to start multi-user session of QAD Enterprise Applications
# tokens:
# &DLC = Progress Directory
# &CLIENT-DB-CONNECT = command line to connect to each db in dbset
stty intr '^c'
DLC=${DLC:-/apps/progress/91d};export DLC
PATH=$PATH:$DLC;export PATH
PROMSGS=$DLC/promsgs;export PROMSGS
PROTERMCAP=$DLC/protermcap;export PROTERMCAP
PS1='$$ ';export PS1
PROPATH=${PROPATH:-.,/mfgsvr/<QXI webapp>,/mfgsvr,/mfgsvr/bbi}
    ;export PROPATH
# Set terminal type.
if [ ${TERM:-NULL} = NULL ]
then
        echo
        echo "Please enter your terminal type: c"
```

```
        read TERM
        export TERM
fi
#
# Start QAD Enterprise Applications.
#
cd # change to home directory
# exec $DLC/bin/_progres &DB etc
exec $DLC/bin/_progres
    /mfgsvr/db/mfgprod -ld qaddb -znotrim -trig triggers
      -db /mfgsvr/db/hlpprod -ld qadhelp
      -db /mfgsvr/db/admprod -ld qadadm
      -d mdy -yy 1920 -Bt 350 -c 30 -D 100 -mmax 3000 -nb 200
      -s 63 -noshvarfix
      -p mfwb01aa.p
      -param mfgwrapper=true,apimode=true
```

PROMSGS

Regardless of the language you are using with your QAD Enterprise Applications installation, the telnet script should reference the US English PROMSGS file. These messages are not normally displayed to the user, but QXI needs to be able to process internal messages correctly.

If you have installed more than one language, ensure the PROMSGS variable points to the US English version.

PROPATH

Add the QXI installation directory in front of the standard PROPATH. In the example below, `/mfgsvr/<QXI webapp>` is the QXI installation directory.

```
PROPATH=${PROPATH:-.,/mfgsvr/<QXI webapp>,/mfgsvr,/mfgsvr/bbi}
    ;export PROPATH#
```

Database Connection Parameters

The sample connection script on page 319 is a local host connection—the QXI and the QAD Enterprise Applications receiver are on the same machine. You can also connect to QAD Enterprise Applications on a separate database server using a client/server connection.

Local Host Connection Parameters

If QXI is on the same server as the QAD Enterprise Applications databases, you can use local host or shared memory connections to access the databases. This can improve QXI performance by eliminating network overhead between the processes and the databases.

To enable local host connections, the QXI parameter files must contain the connection parameters described in Table 25.1.

Table 25.1
Local Host Connection Parameters

Parameter	Description
`-db`	The physical database name parameter. Follow this parameter with the path and physical name of a QAD Enterprise Applications database.
`-trig`	The triggers parameter. Follow this parameter with the name of the directory containing the database triggers for the main QAD Enterprise Applications database. This is the `triggers` subdirectory. The parameter value is: `-trig triggers`. This parameter is used only with main databases.
`-ld`	The logical database name parameter. Follow this parameter with the logical name of a QAD Enterprise Applications database. This parameter is used only with support databases.

Client/Server Connection Parameters

If QXI is on a different machine than the QAD Enterprise Applications databases, use client/server connections to access the databases. This can improve QXI system performance by spreading the QAD Enterprise Applications and QXI resource requirements between two servers.

To enable the server processes to make client/server connections, the QXI parameter files must contain the connection parameters described in Table 25.2.

Table 25.2
Client/Server Connection Parameters

Parameter	Description
`-db`	The physical database parameter name. Follow this parameter with the physical name of the QAD Enterprise Applications database. You do not need to include the path to the database file.
`-trig`	The triggers parameter. Follow this parameter with the name of the directory containing the database triggers for the main QAD Enterprise Applications database. This is the `triggers` subdirectory. The parameter value is: `-trig triggers`. This parameter is used only with main databases.
`-ld`	The logical database name parameter. Follow this parameter with the logical name of an QAD Enterprise Applications database. This parameter is used only with support databases.
`-H`	The host name parameter. Follow this parameter with the machine name or IP address of the QAD Enterprise Applications database server. This guide uses *`DBServer`* as an example in place of an actual machine name or IP address.
`-S`	The database service name parameter. Follow this parameter with a QAD Enterprise Applications database service name. You can use the Database Set Maintenance utility in MFG/UTIL to find the service name for a database.
`-N`	The network parameter. Follow this parameter with the network protocol used to connect to the QAD Enterprise Applications databases. This is TCP/IP. The parameter value is: `-N tcp`.

A Progress on UNIX example of a client/server connection would look like:

```
-db mfgprod -ld qaddb -H svr01 -S prod-srv -N tcp -trig triggers
-db mfghelp -ld qadhelp -H svr01 -S help-srv -N tcp
-db mfgadmin -ld qadadm -H svr01 -S admin-srv -N tcp
```

Additional Parameters

Two additional changes are required for QXI implementation: changing the Progress calling program and adding the Connection Pool Manager parameters. If you are using an HP-UX or Tru64 environment, an additional change may be required.

Progress/Oracle Calling Program

The Progress or Oracle calling program to connect to QAD Enterprise Applications for QXtend is `mfwb01aa.p`. Change the `-p` value to reflect this.

```
-p mfwb01aa.p
```

For details on setting up an Oracle implementation, refer to *Installation Guide: QAD QXtend*.

Connection Pool Manager Parameters

The Connection Pool Manager for QXI requires two additional parameters at the end of the connection script:

```
-param mfgwrapper=true,apimode=true
```

Note For Windows, the connection pool manager parameters should be:

```
-param mfgwrapper=true,apimode=true,DSU=OFF
```

FailureException or AdapterException Responses

In order to avoid a FailureException or AdapterException response, you must modify your connection script or server startup scripts command as defined in the sections below.

HP-UX/COMPAQ True64, Tru64 Environments

If you are operating in an HP-UX, COMPAQ True64 or Tru64 environment, you must add the lines `stty -icanon` and `stty ixoff` to your connection script.

For example:

```
stty -icanon
stty ixoff
SET DLC=c:\dlc91d
SET PATH=%PATH%;%DLC%\bin
SET PROMSGS=%DLC%\promsgs
SET PROTERMCAP=%DLC%\protermcap
```

You also can add these lines to your QXI server startup scripts command line. For example:

```
login:|qxtenduser|Password:|$PASSWD|>|stty -icanon|>|stty ixoff|>|/scripts/qxipool.sh
```

LINUX Environment

If you are operating in a LINUX environment, you must add `| exit` (the pipe symbol and the word `exit`) or `: exit` (a colon and the word `exit`) to your server startup scripts command. For example:

```
login:|qxtenduser|Password:|$PASSWD|>|/scripts/qxipool.sh|exit
```

UNIX Telnet Security

Because QXI communicates over telnet via HTTP, the account log-in ID and password are sent using unencrypted text. Since this may compromise system security, you should configure the telnet environment with server-side security measures in mind.

A range of security options exists to solve the unencrypted log-in and password problem. This section outlines two sample security setups: one providing a maximum level of security and one providing less security but more flexibility for Progress client session and home directory access. In both setups, it is recommended that you use a restricted shell (rsh).

Restricted Shells

Restricted shells are restricted versions of the common UNIX Bourne shell or Korn shell. In the Bourne shell, the restricted shell is run as `rsh` (`/usr/lib/rsh`), while in the Korn Shell it is run as `rksh` (`/usr/bin/rksh`).

The restricted versions of these shells allow users to log in with restricted access. They cannot:

- Use the `cd` command to change directories.
- Specify a path or command using `/`.
- Use redirection (>, >>).
- Set the value of `PATH`.

Note A user's path should not include `/usr/bin`. This lets the user run another shell, thereby inheriting access to any commands that the child shell allows. The default shell for a user is located in the `/etc/passwd` file.

Examples of Security Measures

Case 1: Maximum Security

One UNIX account with the following characteristics is used for all QXI telnet sessions:

- No write permissions to home directory. Temporary files are written elsewhere.
- PATH, DLC, and PROPATH environment variables are set in `.profile` and inaccessible to the user.
- Startup command and/or scripts run from `.profile`.
- Telnet disconnects immediately after the user exits the session.

Use the following instructions to set up Case 1:

1. Create the unique QXI account for log in to UNIX through telnet.
2. Make the default shell for this account the restricted shell.
3. Remove all write permissions for this user in their home directory. Use the `-T` option in the remote script to specify an alternate temporary directory.
4. Set up the `.profile` to set minimal environment variables.
5. Set up the `.profile` to run the script automatically.

Example `.profile` for Case 1:

```
/*Sample .profile for QXtend session, single QXtend login*/
#set default for error (STOP) condition handling
stty intr ^C
#set environment variables
PATH=/dlc91:/dlc91/bin
DLC=${DLC - /dlc91}
PROEXE=${PROEXE - $DLC/bin/_progres}
export PATH DLC PROEXE
#Autorun remote script for QXtend access and automatically exit
exec remote.script
exit
```

Case 2: Less Security

Users have their own unique log in and password, but run the restricted shell by default:

- Write permission to directory is possible, but not necessary.
- PATH, DLC, and PROPATH environment variables are set in `.profile` and inaccessible to the user.
- Users run a subset of UNIX commands, which you add to `/usr/rbin`.
- Users can run the system manually from command line or script.

Use the following instructions to set up Case 2:

1. Create or modify accounts for users of maintenance programs by changing their default shell in the `/etc/passwd` file to the restricted shell.
2. Create the directory `/usr/rbin` and copy the UNIX commands necessary for these users. Make the `/usr/bin` directory read-only so users cannot change path variables.
3. Set up a special `.profile` for the maintenance program users.
4. Set the minimal environment variables, remembering to include `/usr/rbin`.
5. Copy the QXI telnet connection script to each user’s home directory with read-only access.
6. Put any other necessary read-only script files in the home directory.

Example .profile for Case 2:

```
#/* Sample .profile for QXtend session, for individual logins
#remote.script should be in home dir; executable by QXtend*/
#set default for error (STOP) condition handling
stty intr ^C
#set environment variables
PATH=/dlc91:/dlc91/bin:/usr/rbin#don’t forget /rbin directory
DLC=${DLC - /dlc91}
PROEXE=${PROEXE - $DLC/bin/_progres}
export PATH DLC PROEXE
```

Chapter 26

SAX Writer Reference

This section describes the Simple API for XML (SAX) Writer, which is used for creating QDocs inside QAD Enterprise Applications.

Introduction

Extensible Markup Language (XML) is a widely used standard grammar for exchanging business documents between applications, especially over communications networks. QAD Enterprise Applications defines QDocs—XML documents containing QAD Enterprise Applications application data that conforms to a QAD-specified syntax—for this purpose.

Many interoperability scenarios require QDocs or other XML documents to be generated within QAD Enterprise Applications and exported, perhaps as a result of some interesting business event that updates the QAD Enterprise Applications database. The purpose of the SAX Writer API is to provide QAD Enterprise Applications software developers with a high-level set of callable procedures that can be used to create such XML documents within the QAD Enterprise Applications run-time environment, hiding many of the unnecessary details of QDoc and XML syntax from the programmer.

The QDocSAXWriter class supports the following:

- If the QDoc must be created in the 1.0 standard and it contains array fields, the profile dataset is passed in and serialized to XML using the SAX Writer.
- If the QDoc is a delta QDoc, the profile dataset is passed in and serialized to XML using the SAX Writer.

SAX Writer Class

The QDoc SAX Writer class is called `com.qad.xml.QdocSAXWriter`. The SAX Writer class is available for OpenEdge 10.1A and over. The SAX Writer is a stand-alone program that must be started using standard OO4GL constructs. For example:

```
define variable oQdocSAXWriter as class com.qad.xml.QdocSAXWriter.
oQdocSAXWriter = new com.qad.xml.QdocSAXWriter().
```

The QDoc SAX Writer can be used in several ways:

- Create every node in the QDoc manually.
- Create some nodes manually and pass a ProDataSet to populate other nodes.
- Create some nodes manually and pass a buffer handle to populate other nodes.
- Create some nodes manually and include an XML document fragment.
- Pass just the QDoc parameters and a ProDataSet to create the QDoc automatically.

All methods to create an XML document should follow the same structure.

1. Set the XML output method. Use one of the three methods `setOutputToLongChar`, `setOutputToFile`, or `setOutputToMemptr` to specify the output type.
2. Set the QDoc XML syntax. Use the method `setQDocXMLSyntax10` or `setQDocXMLSyntax11`.

   Note Steps 3, 4, and 5 can be combined into one call by using the `buildQdocFromProDataSet` method, or each step can be performed individually.
3. Start the document. Use one of four methods: `startDocument`, `startQdocSOAPEnvelope`, `startQdocSOAPBody`, or `startQdocRequestBody`.

4 Add the nodes. Use the methods `addNode`, `addNamespace`, `addNodesFromBuffer`, `addAllNodesFromBuffer`, or `addNodesFromProDataSet`.

5 End the document. Use the `writeDocument` method.

6 Get the document. Use the `getXMLAsLongChar` and `getXMLAsMemptr` methods.

SAX Writer API Methods

This section describes the primary methods provided by the SAX Writer.

setOutputToLongChar

This method sets the output type of the XML document to be a longchar. This method has no parameters.

setOutputToFile

This method sets the output type of the XML document to be a file. The following table lists parameters of setOutputToFile.

Parameter Name	Type	Description	I/O	Req
`pcFileName`	Character	Name of the file to write the XML to.	I	Yes

setOutputToMemptr

This method sets the output type of the XML document to be a memptr. This method has no parameters.

getXMLAsLongChar

This method gets the XML document as a longchar. The method returns nothing if the output method was not set to be a longchar. Otherwise the method returns a longchar. This method has no parameters.

getXMLAsMemptr

This method gets the XML document stored in a memptr. This method return nothing if the output method was not set to be a memptr. Otherwise the method returns a memptr. This method has no parameters.

setQDocXMLSyntax10

This method sets the XML syntax used to create the QDoc to be 1.0. This method has no parameters.

setQDocXMLSyntax11

This method sets the XML syntax used to create the QDoc to be 1.1. This method can be used only with QAD QXtend version 1.4 and above. This method has no parameters.

buildQdocFromProDataSet

This method creates an entire QDoc, including the SOAP details. The node names in the QDoc are the same names as the buffers and fields in the ProDataSet. This method requires the QDoc syntax and output type to be already set.

The following table lists parameters of buildQdocFromProDataSet.

Parameter Name	Type	Description	I/O	Req
`phDataSet`	Handle	Handle to the dataset with the QDoc data in.	I	Yes
`pcSenderURL`	Character	URL that identifies the sending application.	I	Yes
`pcReceiverURL`	Character	URL that identifies the receiver in QXtend.	I	Yes
`pcConfirmationLevel`	Character	Indicates the confirmation level required (QDoc Syntax 1.0 only).	I	No
`pcMessageID`	Character	Any character ID that identifies this message.	I	Yes
`pcQdocName`	Character	Name of the QDoc that is being created.	I	Yes
`pcQdocVersion`	Character	Version number of the QDoc.	I	Yes
`pcQdocNamespace`	Character	Namespace of the Qdoc	I	No
`plScopeTransaction`	Logical	Identifies whether QXtend will process the entire QDoc as a single transaction.	I	Yes
`plSuppressRespDetails`	Logical	Identifies whether or not QXtend should return a detailed response QDoc.	I	Yes
`plMnemonicsRaw`	Logical	Identifies whether the QDoc contains the mnemonic or the specific language value for translatable values.	I	Yes
`pcDomain`	Character	The domain in QAD Enterprise Applications this data will be loaded in to.	I	No
`pcUsername`	Character	The user name used to log in to QAD Enterprise Applications.	I	No
`pcPassword`	Character	The password for the user.	I	No

startQdocSOAPEnvelope

This method starts a new document and creates a new SOAP envelope node with the correct namespaces defined. This method has no parameters.

endQdocSOAPEnvelope

This method closes the SOAP envelope node. This method has no parameters.

createQdocSOAPHeader

This method controls the process of creating a SOAP header that conforms to the QDoc standards.

The following table lists parameters of createQdocSOAPHeader.

Parameter Name	Type	Description	I/O	Req
pcSenderURL	Character	URL that identifies the sending application.	I	Yes
pcReceiverURL	Character	URL that identifies the receiver in QXtend.	I	Yes
pcDocumentStandard	Character	Identifies the standard that controls the content of the XML in the body (QDoc syntax 1.0 only).	I	No
pcDocumentType	Character	Identifies the specific type of message contained in the request body (QDoc syntax 1.0 only).	I	No
pcDocumentVersion	Character	Identifies the specific version of the message contained in the body (QDoc syntax 1.0 only).	I	No
pcConfirmationLevel	Character	Indicates the confirmation level required (QDoc syntax 1.0 only).	I	No
pcMessageID	Character	Any character ID that identifies this message.	I	Yes
pcDomain	Character	The domain in QAD Enterprise Applications this data will be loaded in to.	I	No
pcUsername	Character	The user name used to log in to QAD Enterprise Applications.	I	No
pcPassword	Character	The password for the user.	I	No
plSuppressRespDetails	Logical	Identifies whether or not QXtend should return a detailed response QDoc.	I	Yes

startQdocSOAPBody

This method creates a new SOAP body node with the correct namespace. This method has no parameters.

endQdocSOAPBody

This method closes the SOAP body node. This method has no parameters.

startQdocRequestBody

This method controls the process of creating the content of the SOAP body. It builds the standard QDoc root node and allows the caller to define all of the settings available on the QDoc root node.

The following table lists parameters of startQdocRequestBody.

Parameter Name	Type	Description	I/O	Req
pcQdocName	Character	Name of the QDoc that is being created.	I	Yes
pcQdocVersion	Character	Version number of the QDoc.	I	Yes
pcQdocNamespace	Character	Namespace of the Qdoc	I	No
plScopeTransaction	Logical	Identifies whether QXtend will process the entire QDoc as a single transaction.	I	Yes
plSuppressRespDetails	Logical	Identifies whether or not QXtend should return a detailed response QDoc.	I	Yes
pcDomain	Character	The domain in QAD Enterprise Applications this data will be loaded in to.	I	No
plMnemonicsRaw	Logical	Identifies whether the QDoc contains the mnemonic or the specific language value for translatable values.	I	Yes
pcUsername	Character	The user name used to log in to QAD Enterprise Applications.	I	No
pcPassword	Character	The password for the user.	I	No

endQdocRequestBody

This method closes the QDoc root node.

The following table lists parameters of endQdocRequestBody.

Parameter Name	Type	Description	I/O	Req
pcQdocName	Character	Name of the QDoc that is being created.	I	Yes
pcNamespace	Character	Namespace of the QDoc	I	No

startDocument

This method starts the a new XML document. This method has no parameters.

startIteration

This method adds a new node to the XML document being created. The following table lists parameters of startIteration.

Parameter Name	Type	Description	I/O	Req
pcIterationName	Character	Name of the iteration to add to the QDoc.	I	Yes
pcNamespace	Character	Namespace of the node	I	No

addNode

This method adds a new node to the XML document being created and writes the data from the passed-in character value.

The following table lists parameters of addNode.

Parameter Name	Type	Description	I/O	Req
pcNodeName	Character	Name of the node to add to the QDoc.	I	Yes
pcNodeValue	Character	Value of the node being added.	I	Yes
pcNamespace	Character	Namespace of the node	I	No

addNamespace

This method adds a new namespace to the XML document being created.

The following table lists parameters of addNamespace.

Parameter Name	Type	Description	I/O	Req
pcNamespaceName	Character	The namespace code.	I	Yes
pcNamespaceURI	Character	The namespace URL.	I	Yes

endIteration

This method closes an iteration node.

The following table lists parameters of endIteration.

Parameter Name	Type	Description	I/O	Req
pcIterationName	Character	Name of the iteration to close.	I	Yes
pcNamespace	Character	Namespace of the node	I	No

addNodesFromBuffer

This method adds a defined set of fields from the current buffer record into the QDoc XML. The caller defines the list of fields to extract from the buffer and a corresponding list of the name that each field’s node should be assigned in the QDoc XML.

The following table lists parameters of addNodesFromBuffer.

Parameter Name	Type	Description	I/O	Req
phBuffer	Handle	Handle to the buffer to extract the fields from.	I	Yes
pcFieldList	Character	Comma delimited list of the field names to extract from the buffer.	I	Yes

Parameter Name	Type	Description	I/O	Req
pcNodeNameList	Character	Comma delimited list of the node names to use for the data extracted from the buffer.	I	Yes
pcNamespace	Character	Namespace of the nodes in the buffer.	I	No

addAllNodesFromBuffer

This method adds all of the fields from the current buffer record into the QDoc XML. The names of the nodes are pulled from the name of the fields in the temp table.

The following table lists parameters of addAllNodesFromBuffer.

Parameter Name	Type	Description	I/O	Req
phBuffer	Handle	Handle to the buffer to extract the fields from.	I	Yes
pcNamespace	Character	Namespace of the nodes in the buffer	I	No

addNodesFromProDataSet

This method controls the process of parsing the contents of the ProDataSet and extracting all of its data into the QDoc that is being generated.

The following table lists parameters of addNodesFromProDataSet.

Parameter Name	Type	Description	I/O	Req
phProDataSet	Handle	Handle to the dataset to extract the data from.	I	Yes
pcNamespace	Character	Namespace of the nodes in the buffer.	I	No

writeDocument

This method closes the document and triggers the writing of the file to the specified output location. This method has no parameters.

insertLogicalAttribute

This method inserts a new attribute into the XML and controls the format of the text so that it is either true or false.

The following table lists parameters of insertLogicalAttribute.

Parameter Name	Type	Description	I/O	Req
pcAttributeName	Character	Name that is assigned to the attribute being created in the XML.	I	Yes
plAttributeValue	Logical	Logical value.	I	Yes

insertDocumentFragment

This method inserts a new XML document fragment.

The following table lists parameters of insertDocumentFragment.

Parameter Name	Type	Description	I/O	Req
`plcFragment`	Longchar	The XML to insert.	I	Yes

Chapter 27

DOM Builder Reference

This section describes the DOM Builder, an API for creating QDocs and other XML documents inside QAD Enterprise Applications.

Note The DOM Builder creates QDocs in the 1.0 syntax specification only.

Introduction

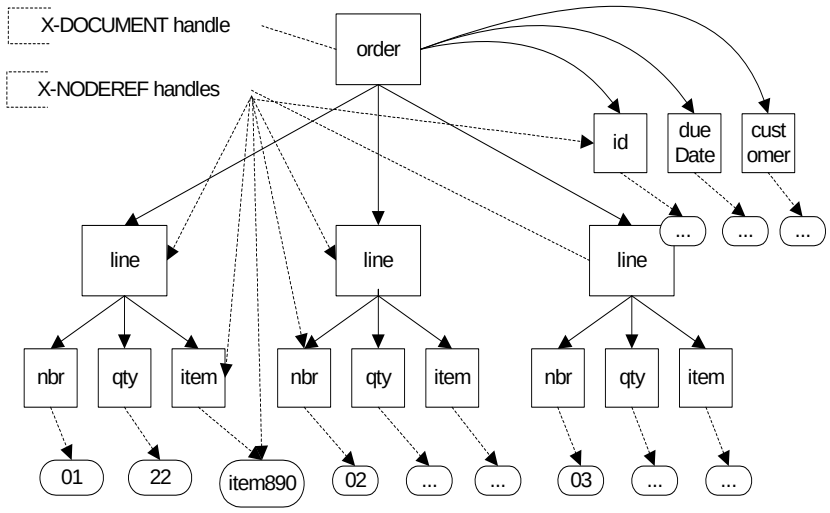
The name DOM Builder comes from the term *Document Object Model (DOM)*, the major industry-standard API for building XML documents. DOM objects are tree-representations of XML documents that contain the parent-child node relationships of XML parent elements, child elements, and attributes and values, letting the software developer navigate from parent nodes to child nodes or vice-versa using designated API methods.

Beginning with Progress 9, the Progress 4GL contains language elements that allow XML documents to be represented and manipulated in memory as DOM objects. These XML objects may be constructed, navigated, and parsed node-by-node using particular Progress 4GL methods. The objects are referenced in Progress source code as X-DOCUMENT and X-NODEREF handles.

The following sample illustrates a simple XML document and its partial DOM representation. Figure 27.1 is a graphical representation of this code sample.

```
<order>
    <id>ord1234</id>
    <dueDate>2003-08-06</dueDate>
    <customer>cust567</customer>
    <line>
        <nbr>01</nbr>
        <qty>22</qty>
        <item>item890</item>
    </line>
    <line>
        <nbr>02</nbr>
        <qty>7</qty>
        <item>item123</item>
    </line>
    <line>
        <nbr>03</nbr>
        <qty>32</qty>
        <item>item456</item>
    </line>
</order>
```

Fig. 27.1
XML Document and DOM Representation



Using the DOM Builder, QAD Enterprise Applications programs can:

- Assemble QAD Enterprise Applications data to be placed into an XML document in either character variables or temp-table form.
- Send the information to the DOM Builder one level or one node at a time.
- Obtain the handle of the resulting XML parent node.

Once completed, the entire XML document can be retrieved as either a handle, temp-table, or character value. The QAD Enterprise Applications developer does not have to directly manipulate each XML node using the lower-level XML methods of Progress, as the DOM Builder does so automatically.

DOM Builder Procedures

The DOM Builder is a group of API methods written as internal procedures in two QAD Enterprise Applications programs that are run persistently.

- General XML DOM Builder: `gpdomcr.p`
- QDoc DOM Builder: `gpqdoccr.p`

The XML DOM Builder is a super-procedure of the QDoc DOM Builder, as the latter reimplements some of the general XML methods to automatically add attributes that appear in all valid QDocs, such as the QAD-defined XML namespaces. The QDoc DOM Builder should be started as persistent from QAD Enterprise Applications if the objective is to generate QDocs. It will automatically launch the XML DOM Builder.

If the objective is to generate non-QDoc XML, the general XML DOM Builder should be started as persistent instead. Most of the methods are applicable to both QDocs and other XML documents, but some of the methods are QDoc-specific. See “QDoc-Specific API Methods” on page 341.

Most programs that use the DOM Builder API would structure their method calls as follows:

- Start `gpqdoccr.p` or `gpdomcr.p` running persistently, depending on whether the intent is to create QDocs or other kinds of XML documents.
- Call `createNewXMLDocument` to create the XML document.
- Call `addNode`, `addNodeGroup`, `addRecordNode`, `addAttributes`, and `createException` in combination in order to populate the XML document level-by-level, node-by-node.
- Call `getXMLAsTempTable`, `getXMLAsDOM`, `getXMLAsString`, or `getXMLAsFile` to retrieve the completed XML document objects in the form required.
- Call `deleteDocument` to delete the XML objects.

Important Only a single XML document can be under construction at one time in a single QAD Enterprise Applications session. Each time `createNewXMLDocument` is called, any XML document already in process for that session is lost and a new one started.

DOM Builder and Q/LinQ

The DOM Builder is a general-purpose API that is not part of Q/LinQ. However, it can easily be used in conjunction with the Q/LinQ publishing API to publish QDocs and other XML documents for outside applications. In Q/LinQ publishing scenarios, the DOM Builder methods would be

called to create the desired XML object and serialize it into the records of a temp-table. The resulting temp-table would be passed to the `qqPublishXMLDoc` method of the publishing API. See *Technical Reference: QAD Q/LinQ* for details on Q/LinQ APIs.

The following partial code sample illustrates how the two APIs can be used together. Also, the Q/LinQ program responsible for creating QDoc confirmation documents, `qqsndack.p`, is a good working example.

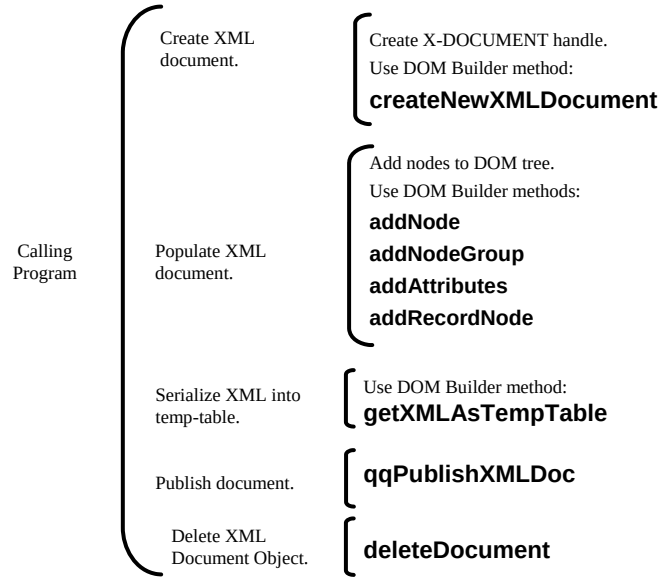
Use of the publishing API consists of a single call to `qqPublishXMLDoc`, but requires that the XML document be built beforehand using the XML capabilities of the Progress 4GL. It accepts an X-DOCUMENT handle to an XML document, and serializes it into the Q/LinQ export queue.

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

{pxrun.i
    &PROC = createNewXMLDocument
    &PARAM = “(
        ...)”
    &HANDLE = domBuilder
    &CATCHERROR = true
    }

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

...

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

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

    …
{pxrun.i
    &PROC = qqPublishXMLDoc
    &PARAM = “(
        ...)”
    &HANDLE = domBuilder
    &CATCHERROR = true
    }

/*Delete the completed document to reclaim memory resources*/
{pxrun.i
    &PROC = deleteDocument
    &HANDLE = domBuilder
    &CATCHERROR = true
    }
```

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

Figure 27.2 illustrates the logic of publishing in XML format.

Fig. 27.2
Publishing API Adapter Logic for XML Documents



QDoc-Specific API Methods

This section describes the primary methods provided by the DOM Builder that are applicable only to QDocs.

createException

This method processes all exceptions present in the `tmp_err_msg` table, creating a QDoc exception element for each record and adding it to the in-process QDoc under the given parent element node. It is useful for building a QDoc response for an earlier QDoc request.

The `tmp_err_msg` temp-table is used by some QAD Enterprise Applications programs, and in particular by all QAD Enterprise Applications API procedures, to contain QAD Enterprise Applications error and warning messages generated during transaction processing. It is defined in the include file `pxttmsg.i`.

The following table lists parameters of `createException`.

Parameter Name	Type	Description	I/O	Req
`errorTableBuffer`	Handle	Handle to the `tmp_err_msg` temp-table buffer.	I	Y
`parentNode`	Handle	`X-NODEREF` object pointing to the parent node under which the exception elements will be placed as child elements.	I	Y

createNewXMLDocument

This method creates a new QDoc.

The following table lists parameters of `createNewXMLDocument`.

Parameter Name	Type	Description	I/O	Req
`qdocName`	Char	Name of the QDoc root element.	I	Y
`qdocVersion`	Char	Version attribute of the QDoc root element.	I	Y
`qdocSchemaLocation`	Char	URL of the QDoc's XML schema, for syntax validation purposes.	I	N
`sxopeTrans`	Logical	Indicates whether the entire QDoc must be processed as a single unit of work.	I	Y
`logTrans`	Logical	Indicates whether the QDoc should be logged during processing.	I	Y
`transId`	Char	User-defined identifier for the QDoc, for tracking purposes if it is logged.	I	N
`suppressResponseDetail`	Logical	Indicates whether the response to the QDoc should include detailed application data, or only the outcome of processing with any exceptions.	I	Y
`qdocRootNode`	Handle	`X-NODEREF` object pointing to the root node of the new QDoc.	O	Y

createReturnStatus

This method creates a node in the QDoc indicating the success or failure of an earlier corresponding QDoc request. It is useful for building a QDoc response for an earlier QDoc request.

The following table lists parameters of `createReturnStatus`.

Parameter Name	Type	Description	I/O	Req
`returnStatus`	Char	Value to be stored as the return status of an earlier request.	I	Y
`qdocNode`	Handle	`X-NODEREF` object pointing to the QDoc root node.	I	Y

createSOAPMessage

This method creates a SOAP 1.0 envelope with a valid QDoc SOAP header, and wraps the designated QDoc into the SOAP body. It is useful for packaging a QDoc in the form that would be required for a Web service request or response.

The following table lists parameters of `createSOAPMessage`.

Parameter Name	Type	Description	I/O	Req
`senderURL`	Char	URL that identifies the sending party or application.	I	N
`receiverURL`	Char	URL that identifies the receiving party or application.	I	N
`senderDocumentId`	Char	Locally unique identifier for the QDoc SOAP message being generated.	I	N
`documentStandard`	Char	Identifies the standard or specification to which the QDoc content conforms. Normally set to qdoc.	I	N
`documentType`	Char	Identifies the type of QDoc being created. Normally set to the QDoc root element name.	I	N
`documentVersion`	Char	Identifies the version of the QDoc being created. Normally set to the QDoc version attribute.	I	N
`confirmationLevel`	Char	Indicates whether or when a response to the created QDoc-based message is required at some later time, for confirmation purposes. Normally set to all, error, or none.	I	N
`senderDocumentRef`	Char	Reference to a business or data object in the sending application.	I	N
`receiverDocumentRef`	Char	Reference to a business or data object in the receiving application.	I	N
`timeZone`	Char	Time zone of the system creating the request, in the format of the QAD Enterprise Applications Time Zone field (tzo_mstr.tzo_label) maintained in Multiple Time Zones Maintenance (36.16.22.1) or the Time Zone field (qqc_ctrl.qqc_timezone) maintained in Q/LinQ Control (36.8.24).	I	N
`requestRootNode`	Handle	`X-NODEREF` object pointing to the root element node of the QDoc to be included in the SOAP message.	I	Y

General XML API Methods

This section describes the primary methods provided by the DOM Builder that are applicable to all XML documents, whether QDocs or not.

addAttributes

This method adds a list of attributes and the values to the specified XML element node.

Important This method is implemented as a Progress function, not an internal procedure.

Return Codes

Type	Description
Logical	Indicates whether the function call was successful.

The following table lists parameters of `addAttributes`.

Parameter Name	Type	Description	I/O	Req
`currentNode`	Handle	Handle of the element node to which the attributes will be added.	I	Y
`attributeNames`	Char	List of the attribute names, delimited by the ASCII 3 (Progress chr(3)) character.	I	Y
`attributeValues`	Char	List of the attribute values, delimited by the ASCII 3 (Progress chr(3)) character. The number of entries is the same as the attributeNames list, and the order of values is the same as the order of names.	I	Y

addNode

This method creates a new child element node under a given parent element node, with a specified name and value.

The following table lists parameters of `addNode`.

Parameter Name	Type	Description	I/O	Req
`nodeName`	Char	Name of the new element node.	I	Y
`nodeValue`	Char	Value of the new element node.	I	N
`parentNode`	Handle	`X-NODEREF` object pointing to the element node that will be the parent of the new element.	I	Y
`newNode`	Handle	`X-NODEREF` object pointing to the new element node.	O	Y

addNodeGroup

This method creates a group of element nodes and adds them as children under a given parent element node, each with a specified name and value.

The following table lists parameters of `addNodeGroup`.

Parameter Name	Type	Description	I/O	Req
`nodeNames`	Char	List of names of the new element nodes, delimited by the ASCII 3 (Progress chr(3)) character.	I	Y

Parameter Name	Type	Description	I/O	Req
nodeValues	Char	List of values of the new element nodes, delimited by the ASCII 3 (Progress chr(3)) character. The number of entries is the same as the nodeNames list, and the order of values is the same as the order of names.	I	N
parentNode	Handle	X-NODEREF object pointing to the node that will become the parent element of the new elements.	I	Y

addRecordNode

This method creates an element node with a group of child element nodes under a given parent element, where the group of created nodes is an XML representation of a buffer. The caller can specify which fields from the buffer to include in the XML or, alternatively, which fields to exclude.

The following table lists parameters of addRecordNode.

Parameter Name	Type	Description	I/O	Req
nodeName	Char	Name to give the top-level element node to be created, under which element nodes corresponding to the fields in the buffer will be added as children.	I	Y
bufferHandle	Handle	Handle of the buffer whose contents will be used to create child element nodes.	I	Y
prefixToStrip	Char	A fixed prefix of characters (for example, a database table identifier) to be stripped from the XML element names derived from the buffer field names.	I	N
includeMode	Logical	Indicates whether the caller-provided list of buffer fields represent data to include in or exclude from the resulting XML.	I	N
fieldList	Char	A list of field names from the buffer that will be included/excluded from the resulting XML, depending on the includeMode parameter. The list is delimited by the ASCII 3 (Progress chr(3)) character.	I	N
parentNode	Handle	Handle to the element node under which the generated top-level element nodes will be placed as a child, and the elements corresponding to the buffer fields will be placed as grandchildren.	I	Y
newNode	Handle	Handle to the top-level node created.	O	Y

createNewXMLDocument

This method creates a new XML document.

To create a QDoc, use the QDoc-specific version of this method. See page 342.

The following table lists parameters of createNewXMLDocument.

Parameter Name	Type	Description	I/O	Req
`initialXml`	Char	Valid XML string used as the starting point of the new XML document.	I	N
`rootNode`	Handle	`X-NODEREF` object pointing to the root node of the new XML document.	O	Y

deleteDocument

This method deletes all the XML document and node objects that have been created by the DOM Builder. It should be called by any programs using the DOM Builder after it has finished building an XML document, in order to reclaim memory and eliminate the possibility of referencing stale handles later on.

The following table lists parameters of `deleteDocument`.

Parameter Name	Type	Description	I/O	Req
None				

getXMLAsDOM

This method returns the node referencing the generated XML document.

The following table lists parameters of `getXMLAsDOM`.

Parameter Name	Type	Description	I/O	Req
`builtDocument`	Handle	`X-DOCUMENT` node pointing to the generated XML document.	O	Y

getXMLAsFile

This method serializes the content of an XML document object into ASCII characters and stores it in a designated text file.

The following table lists parameters of `getXMLAsFile`.

Parameter Name	Type	Description	I/O	Req
`xmlFile`	Char	Name of the text file in which the generated XML document will be saved.	I	Y

getXMLAsString

This method serializes the content of an XML document object into a single ASCII character string.

Important Due to Progress size limitations, the number of characters in the XML document cannot exceed 32,000. If the XML document could be larger than that, it should be retrieved using the `getXMLAsTempTable` or `getXMLAsFile` method instead.

The following table lists parameters of `getXMLAsString`.

Parameter Name	Type	Description	I/O	Req
`xmlString`	Char	Text of the generated XML document.	O	Y

getXMLAsTempTable

This method serializes the content of an XML document object into ASCII characters and stores it in the records of the `ttXMLDocument` temp-table. The document is partitioned into 15,000-character chunks, with each chunk stored in a single record of the temp-table.

Important The `ttXMLDocument` temp-table is defined by include file `gpdomcr.i`. This file must be included in any QAD Enterprise Applications program that intends to load the finished XML document into a temp-table using the `getXMLAsTempTable` method. The include file defines the temp-table used by the DOM Builder to pass the XML data back to the caller.

The following table lists parameters of `getXMLAsTempTable`.

Parameter Name	Type	Description	I/O	Req
`ttXMLDocument`	Temp-table	Temp-table that will contain the XML data in chunks of 15,000 characters.	I	Y

Utility Methods

In addition to the above methods, the following utility methods are also available in `gpdomcr.p`. They perform various lower-level data manipulation functions useful in the generation of XML, such as converting QAD Enterprise Applications dates and languages into the standard ISO syntax required for most XML-based standards. For detailed information, see the `gpdomcr.p` source code.

- convertDataFormat
- createNode
- getDate
- getDateTime
- getLogical
- getXMLSchemaType
- getXMLLanguage
- processArrayField
- stripPrefix

Chapter 28

QAD QXtend Exception Codes

This section provides descriptions of the QXtend exception codes.

Overview

This section contains a complete list of possible QAD QXtend Inbound (QXI) exceptions. Exceptions can include other exceptions. An included exception is contained in the response QDoc (unless `excludeTraces` is set) and in the log files (regardless of `excludeTraces`); the included exception can provide more insight into the problem. See “Include Java Trace Information” on page 155 for information on the excludeTraces setting.

Table 28.1 provides the directory locations of the log and configuration files referenced in the exception codes. See “Logging” on page 158 for a discussion of all the QXI logging options.

Table 28.1
Configuration and Log Files by Location

Path	Files
TOMCAT_HOME\webapps\<*QXI webapp*>\qxtendQueues	directoryloaderconfig.xml soapconfig.xml
TOMCAT_HOME\webapps\<*QXI webapp*>\qxtendQueues*QUEUE_NAME*	queueconfig.xml
TOMCAT_HOME\webapps\<*QXI webapp*>\WEB-INF\conf	config-files.xml connectionManagerConfig.xml environmentmanager.xml qxtendconfig.xml routings.xml
TOMCAT_HOME\webapps\<*QXI webapp*>\WEB-INF\descriptors*MFGPRO_VERSION*	implementationQdocs.xml qadQdocs.xml (global)
TOMCAT_HOME\webapps\<*QXI webapp*>\WEB-INF\logs	connectionPools.log qdocInfo.log queue.log.*date* qxtendServer.log
TOMCAT_HOME\webapps\<*QXI webapp*>\WEB-INF\receivers	qdocReceivers.xml
TOMCAT_HOME\webapps\<*QXI webapp*>\WEB-INF\receivers*RECEIVER_NAME*\descriptors	implementationQDocs.xml qadQdocs.xml (receiver specific)

Transformation Exceptions

Transformation exceptions are raised by the QXI transformation engine when applying transformations to documents that do not conform to QXI standards. Transformation exception codes use the format “TransformationCode” plus the code identifier—for example, TransformationException001.” The Exception Code column in Table 28.2 shows only the code identifier.

Table 28.2
License Exceptions

Exception Code	Description	Notes
001	Specified key does not exist in myParserClass hashmap	This configuration issue should never occur. When transformation has been configured, the `requestparsermanager.xml` file is used to control the creation of RequestParsers that are used to parse the request to extract information from the request. If this exception occurs, the requested RequestParser type has not been configured in the `requestparsermanager.xml` file.
002	Could not create specified class	The transformation engine uses Reflection—which creates classes at runtime based off parameters—to create instances of classes with the type specified in a configuration file. If the process of creating a class fails, this exception is raised. This exception means that the class specified in the configuration file `TransformationManager.xml` or the RequestParser manager could not be instantiated.
003	Could not find the `requestParserManager.xml` file	The `requestparsermanager.xml` file must be located in the `<TomcatHome>/webapps/<qxtend-web-app>/WEB-INF/conf` directory. This error indicates that the file cannot be found.
004	Problem occurred while creating DocumentWrapper	Indicates the XML document being loaded is invalid.
005	Specified key does not exist in myTransformationClasses	Similar exception to 001. However, instead of not having the RequestParser configured, it is the Transformer class that has not been configured in the `transformationmanager.xml` file.

Exception Code	Description	Notes
006	Could not locate the `transformationmanager.xml` file at the specified location	The `transformationrmanager.xml` file must be located in the `<TomcatHome>/webapps/<qxtend-web-app>/WEB-INF/conf` directory. This error indicates that the file cannot be found.
007	No transformer node in `transformationmanager.xml`	The `transformationmanager.xml` file requires a predefined structure. As part of that structure it requires a node named `transformer`. If that node cannot be found in the correct place, this exception is thrown.
008	Envelope header is null	When processing transformations it is required that the request contains a SOAP envelope, this error is thrown if a request does not contain a SOAP envelope.
009	Problem getting request header and body	When processing Transformations the engine split the SOAP message into two parts—the header and the body. This exception indicates that the request could not be split.
010	Problem occurred creating transformer	Transformation uses a shared library for executing the transformations. The first step is to obtain a Transformer class from a TransformerFactory class. This exception is thrown when the creation of the Transformer class fails.
011	Error occurred during transformation	An unhandled error occurred while the transformer was processing the XSLT transformation.
012	Problem occurred while setting the response header and body	While building the response message—which is a SOAP message with a header and a body—an unhandled error occurred.
013	No child node of root node present	If the request XML being transformed does not have a root node (single node with no parent), this exception is thrown.
014	Problem getting parser from requestParserManager	The Request Parser Manager controls access to all request parser instances. This exception is raised when a request parser is requested from the Manager, and it cannot return one.
015	Could not find the queue configuration file	Indicates that for a specific queue the `queueconfig.xml` file could not be found.

Exception Code	Description	Notes
018	No `requestParsers` node present in XML	The `requestparsermanager.xml` file requires a predefined structure. As part of that structure it requires a node named `requestParsers`. If that node cannot be found in the correct place, this exception is thrown.
019	No matching mapping specification found	The valid mapping specifications are defined in `transformationmanager.xml`. This exception indicates that for the combination of documentStandard, documentType, documentVersion, and receiver that no mapping specification could be located to use to transform the message. Change the configuration in `transformationmanager.xml` to ensure that there is a mapping specification defined.

Event Exceptions

Events are responsible for assisting in the navigation of the QAD Enterprise Applications screens. These exceptions may occur during the processing of a QDoc. The event exceptions are visible in the response QDoc and in the logs.

Table 28.3
Event Exceptions

Exception Code	Description	Notes
EventException001	Error processing data	During the processing of a QDoc request, QXtend passed through the same field twice that was not the start of an iteration. The field in question can be found in the exception detail.
EventException002	Unexpected prompt field after delete	Unexpected field name in the events file for the delete confirmation prompt. The expected field name is usually the response to the QAD Enterprise Applications confirmation query: Are you sure you wish to delete this record? Review the events file or use QGen to update the file.
EventException003	Error while deleting	An QAD Enterprise Applications error occurred while deleting a record. Review the data in the QDoc and ensure that this data can be deleted.
EventException004	Unable to load an events document	The specified events file cannot be found or loaded. There may be invalid XML in the file.

Exception Code	Description	Notes
EventException005	Error creating an event object	While adding a new events type, no Java class exists with the event name. Check the events file for the event types. This occurs only if the events file has been incorrectly updated. Review the events file for an invalid event type such as IterationEvent: `<IterationEvent iterationname= "supplier" exititeration="f4"/>`
EventException006	Object is not an event	The event object created (see EventException005) does not conform to a specific Java interface. This occurs only if the events file has been incorrectly updated. Review the events file for an invalid event type such as IterationEvent.
EventException007	Incorrect iteration	Internal exception that arises when QXtend is attempting to resynchronize with the UI after an unexpected UI event.
EventException008	No name found for the iteration	Current field at the start of each iteration cannot be located in the events file. Review the events file for this program; it may need remapping using QGen.
EventException009	Invalid selection in selection list	The value entered in the QDoc for a selection list is not in QAD Enterprise Applications. Review the data in the QDoc and ensure that the selection item exists in QAD Enterprise Applications.
EventException010	No data in the QDoc	No data to process in the QDoc.
EventException011	Data sent too large	The data submitted for a primary field is too large. It may update an existing record rather than create a new record. Review the data in the QDoc.
EventException012	Ambiguous selection in selection list	Data for a selection list in the QDoc exists in more than one row in QAD Enterprise Applications. Review the data in the QDoc and QAD Enterprise Applications for this selection list.
EventException013	The QDoc has processed successfully but some QAD Enterprise Applications messages may have been lost	Occurs when switching from reading messages using the MessageReceiverServlet and the message area of the UI.
EventException014	Preprocess program name is null	The program name for the preprocess event is null. Verify the name in the events file.
EventException015	Error calling preprocess program	An error occurs while calling the preprocessor program. Review the events file.

Exception Code	Description	Notes
EventException016	Preprocess – Updated iteration node does not match existing iteration node	The preprocess program changes the name of the current iteration. This is invalid. Correct the preprocess program code.
EventException017	Postprocess – Updated response iteration node does not match existing response iteration node	The postprocess program changes the name of the current iteration. This is invalid. Correct the postprocess program code.
EventException018	An attempt was made to access invalid events data. See context for details.	The events data file does not match the fields in the QDoc.
EventException019	An error was encountered when updating a node using the QXI preprocessor or postprocessor.	

Connection Exceptions

These exceptions are related to errors raised when using the Connection Pool Manager. These exceptions are included in the response QDocs.

Table 28.4
Connection Exceptions

Exception Code	Description	Notes
ConnectionException001	Unable to initialize Connection Pool	Cannot initialize a connection pool session. The maximum number of sessions has been reached, or the QAD Enterprise Applications databases are not up. There may also be a problem with the log-in script or a step omitted. Test this by following the log-in script from the Connection Pool setup parameters and ensuring all steps are correct. Another possible cause is that the startup script for launching the QAD Enterprise Applications session (invoked by the Connection Manager) may have been configured to run in normal mode instead of MFGWrapper mode. (See “Starting QGen” on page 220 for more on the MFGWrapper mode.) Review the startup script and update if incorrect, then restart the Connection Pool.
ConnectionException002	Unable to get a working session using unique IDs (session ID and process sequence ID)	Unable to get a working session using unique IDs.
ConnectionException003	Failure to initialize the Connection Pool Manager	The Connection Pool Manager cannot initialize. One responsibility of the Connection Pool Manager is to open XML configuration files. This error can occur if there is an invalid configuration file. To see a list of configuration files look at `config-files.xml`.
ConnectionException004	Unable to get an idle session	A client requested a new program to be run and an idle session cannot be retrieved from the connection pool. Verify that the connection pool has not exceeded the maximum number of connections. It is also possible that the Connection Pool Manager was initializing new sessions that were not yet available.

Exception Code	Description	Notes
ConnectionException005	Unable to get session because pool has been shut down	A user attempted to create a new session or make a request on an existing one after the Connection Pool Manager was shut down. In this case, all connections are invalid and the Connection Pool Manager must be restarted.
ConnectionException006	User has reached max sessions allowed	A user has requested to run another QDoc; however, this user has exceeded the maximum number of allowed connections.
ConnectionException007	Extracting a value from the configuration file failed	Invalid value encountered during load of `connectionManagerConfig.xm`. Review all numeric values in the file.
ConnectionException008	Unable to locate the Connection Pool Manager configuration file	This file should be located in `webapps/<QXI webapp>/WEB-INF/config`. If it is not there, then a default version of the file can be copied and configured. This is located in `webapps/<QXI webapp>/WEB-INF/config/defaults`
ConnectionException009	Telnet connection not available	The telnet server for the specified receiver is unavailable.
ConnectionException010	Host is unavailable or incorrect	View the connection pool details for this receiver and validate the host is correct and available.
ConnectionException011	Telnet connection is refused	The connection to the host was refused. Manually launch a telnet connection from the QXtend machine to the host machine to isolate the problem.
ConnectionException012	QAD Enterprise Applications server is unreachable	The connection to the host is successful but the launch of an QAD Enterprise Applications session is not. Manually launch a telnet connection from the QXtend machine to the host machine and run the QXtend startup script to isolate the problem.

Queue Exceptions

If an exception is raised during QXtend startup, the UI paints the Queue link in red. This indicates an error that is logged to `queue.log` and `qdocInstall.log`. If an exception is raised during the processing of a QDoc through a queue, this exception appears in the `qdocinfo.log` and possibly the `qxtendserver.log`, as well as in the response QDoc.

Table 28.5
Queue Exceptions

Exception Code	Description	Notes
QueueException001	Unable to create queue object components	Class file names required to determine what Java classes to create are incorrect in `directoryloaderconfig.xml`.
QueueException002	Configuration file does not contain a queue node	The queue configuration file is missing a queue node. Update or create a new queue in the Queue Manager.
QueueException003	The parameter types do not exist as classes	There is no <queue> node in the `queueConfig.xml` file.
QueueException004	Error due to incorrect URL	The URL specified by the user in Update Queue in the Queue Manager is incorrect. The URL is the location of the QXtend Web services component.
QueueException005	Could not load Queue Manager config file	Verify the correct path and existence of: `directoryloaderconfig.xml`.
QueueException006	Could not create class	See QueueException001.
QueueException007	Could not locate `queueConfig.xml`	The queue configuration file has been deleted, moved, or renamed. Update or create a new queue in Queue Manager. Verify correct path and existence of `queueConfig.xml`.
QueueException008	No queue type in queue configuration file	QXI has one valid queue type—qdoc—in Queue Configuration. This exception is raised if the queue type is set incorrectly.
QueueException009	No queueClass element in configuration file	The settings for all queues are stored in an XML configuration file. This file must have been manually updated incorrectly. Using the Queue Manager, delete and recreate the queue to correct this.
QueueException010	No URL element in configuration file	The URL is set incorrectly in the Queue Configuration.
QueueException011	No frequency element in configuration file	The Frequency is set incorrectly in the Queue Configuration.
QueueException012	Base directory does not contain any queues	No queues exist in the base directory. Create a new queue using the Queue Manager.
QueueException013	Envelope details missing from configuration file	SOAP envelope details required for QDoc updates are missing. This only impacts queues configured to add SOAP envelopes to the QDocs placed in the request directory. Update or create a queue.
QueueException014	No QDoc nodes in the default envelope	See QueueException013 for details. Update the queue.

Exception Code	Description	Notes
QueueException015	Queue configuration file missing	The queue configuration file, `queueConfig.xml`, has been deleted, moved, or renamed. Update or create a queue.
QueueException017	Specified queue missing	The queue directory has been removed or renamed. Create a new queue.
QueueException018	Queue Manager configuration file missing	The Queue Manager config file, `directoryloaderconfig.xml`, is missing. This file contains global configuration details.
QueueException019	QueueType node missing from configuration file	QueueType identifies the type of the requests on the queue. Update or create a queue.
QueueException020	No queueType or blank queueType specified in queue configuration file	The Queue Manager config file is incorrect. Open `directoryloaderconfig.xml`. Copy in the XML node shown under “Queue Type Node” following this table.
QueueException021	No queueTypes set up in queue config file	See QueueException020.
QueueException022	Queue Manager base directory missing	The full path to the top-level queue directory must be specified in `environmentmanager.xml`.
QueueException029	The configuration file does not contain the pooler node	No pooler node in `queueConfig.xml`.
QueueException030	The configuration file does not contain the `initialSize` element under the pooler node	No pooler node in the `queueConfig.xml` file.
QueueException031	The configuration file does not contain the `maxSize` element under the pooler node	No pooler node in the `queueConfig.xml` file.
QueueException032	The configuration file does not contain the `webServiceTimeout` element under the pooler node	No pooler node in the `queueConfig.xml` file.
QueueException033	The configuration file does not contain the `retryTime` element under the pooler node	No pooler node in the `queueConfig.xml` file.
QueueException034	The pooler node in the config file contains child elements that specify invalid numbers	An element value should contain integer values in the `queueConfig.xml` file.

Queue Type Node

The following queue type node applies to QueueException020.

```
<queueTypes>
    <!-- Each queue Type supported by the directory loader must be specifed here. Each type
    then has a  processor class and a queue class associated with it-->
    <queueType>
        <!-- type - Identifies the type of a queue.
        processorClass - This is the class that will be used to process any requests on a
        queue of this type
        queueClass - This is the class that will be created to hold a queue of this type -->
        <type value="qdoc"/>
        <processorClass value="com.qad.qxtend.utils.QdocFileProcessor"/>
        <queueClass value="com.qad.qxtend.utils.QdocDirectoryQueue"/>
    </queueType>
</queueTypes>
```

SOAP Exceptions

These exceptions relate to the SOAP messages such as calling a SOAP service or extracting QDocs from the SOAP message. These exceptions appear in the response QDoc.

Table 28.6
SOAP Exceptions

Exception Code	Description	Notes
SOAPException001	Incorrect destination URL for SOAP request	The destination URL that points to the QXI server is invalid or the URL is unreachable.
SOAPException003	SOAP Manager configuration file is missing	Verify the path and existence of `soapconfig.xml`.
SOAPException004	XML document create failed	Invalid XML in `soapconfig.xml`. Review and correct the file.
SOAPException005	Invalid data in SOAP configuration file	Invalid data in `soapconfig.xml`. Review and correct the file.
SOAPException006	Problem returning response	Writing QDoc response file to the SOAP response failed. Likely cause is that the HTTP connection has abnormally closed. View `queue.log`.
SOAPException007	Request from document processor to pass to destination failed	Retrieving the QDoc response file from the SOAP request failed. Likely cause is that the HTTP connection has abnormally closed. View `queue.log`.
SOAPException008	No SOAP configuration file specified	When launching QXtend, if the Queue Manager is active, there is an entry in `environmentmanager.xml` for the SOAP caller manager. One of the nodes is the path to `soapconfig.xml`. Verify the path. View `qdocInstall.log`.
SOAPException009	Server is suspended	QXtend is suspended, possibly as a result of someone updating the configuration. Re-activate using Resume in QXtend.

Transaction Exceptions

Transaction exceptions relate to the QXtend transaction unit, which is made up of the request and response QDocs and manages the process of routing the message through the UI API adapter or service interface adapter. These exceptions appear in the response QDoc.

Table 28.7
Transaction Exceptions

Exception Code	Description	Notes
TransactionException001	Error during processing. Full transaction roll back.	QDoc processing failed. The entire transaction rolls back.
TransactionException002	Invalid routing ID	An invalid routing ID was used in the QDoc. The routing ID determines if the QDoc is processed using the service interface API or UI API. Check the routing for each QDoc in the Configuration Manager under Schemas by QAD Enterprise Applications version.
TransactionException003	Unable to load routing document	The correct document is not in `routing.xml`, or is an invalid XML document.
TransactionException004	Error creating RoutingTask object	RoutingTask object cannot be instantiated. Review `routing.xml` for the QDoc for a particular route.
TransactionException005	RoutingTask object is not a routing task	The RoutingTask object in `routing.xml` is not the correct type.
TransactionException006	Null field returned by the processor	QAD Enterprise Applications reports that the current field is null, or the session that started the program failed to initialize, or QAD Enterprise Applications aborted abnormally. To determine the cause, run an QAD Enterprise Applications CHUI session manually using the Connection Pool Manager startup script and follow the processing in the QDoc.
TransactionException007	Error during processing. Partial transaction roll back.	QXtend was unable to complete the processing of the QDoc. View the response QDoc for the error that caused the problem. This rolls back the current transaction (main iteration level) in QAD Enterprise Applications.

Exception Code	Description	Notes
TransactionException009	Requested delete operation did not occur	QXtend was unable to delete a record in QAD Enterprise Applications.
TransactionException010	Program did not initialize	The QAD Enterprise Applications session failed to initialize; this failure was probably due to an incorrect setting on the QAD Enterprise Applications side. Check for PROGRESS errors and verify that the compilation is correct. Also check the PROPATH settings on the telnet script.

Adapter Exceptions

This exception traps any undiagnosed exceptions during the processing of a QDoc using the UI API adapter. These exceptions appear in the response QDoc.

Table 28.8
Adapter Exceptions

Exception Code	Description	Notes
AdapterException003	Unknown exception servicing QDoc request	An undiagnosed exception was returned during QDoc processing. Review the response details for an indication of the problem. This exception should never occur. Contact QAD Support. For details about avoiding AdapterException responses in HP-UX, COMPAQ, or LINUX environments, see “FailureException or AdapterException Responses” on page 322.
AdapterException004	Exception encountered when fill in foreign keys to response	Exception encountered when fill in foreign keys to response.

QDoc Exceptions

These exceptions report errors in the contents of request or response QDocs. This section also contains exception details when using the Queue Manager to process QDocs. These exceptions appear in the response QDoc.

Table 28.9
QDoc Exceptions

Exception Code	Description	Notes
QDocException001	Request QDoc missing SOAP envelope	Cannot read the QDoc SOAP envelope. Ensure the SOAP envelope is correct and validate the QDoc.
QDocException002	QDoc receiver is not recognized	Ensure the QDoc is set up for the correct receiver and the receiver is set up correctly.

Exception Code	Description	Notes
QDocException003	Request does not contain a request body	QDoc must have a body. Review and correct the QDoc.
QDocException004	Failed to create a new XML document instance	Loading the QDoc schema file failed because the schema does not exist or the schema file does not contain valid XML. Check the schema version in the QDoc and validate the schema file.
QDocException005	QDoc is not supported	Ensure the QDoc is set up for the correct receiver and the receiver is set up correctly; add the appropriate schema to the receiver.
QDocException007	Failed to create a clone of the QDocKey instance	A clone of the QDocKey instance could not be created.
QDocException008	Error creating response QDoc	QXtend cannot create a QDoc response. Usually due to invalid XML data.
QDocException009	Failed to create a clone of the QDoc envelope template	A clone of the QDoc envelope template could not be created.
QDocException010	Failed to create a new QDoc response instance	See QDocException008.
QDocException011	Failed to parse the XML schema	See QDocException004.
QDocException012	Could not read schema file *fileName*	See QDocException004.
QDocException013	Could not set the response object in the transaction instance. Another adapter may have set the response already. The response should be updated.	Update the response.
QDocException014	Failed to locate an initialized Connection Manager instance	The Connection Pool Manager is not initialized correctly. This should only occur on the first use of QXtend. Review Connection Manager, `connectionPools.log`, and `qdocInstall.log`.
QDocException015	Timed out waiting for idle AppServer connection	This applies to the service interface API adapter. Not enough application servers are available for the number of requests. Review the connections in the Connection Pool and update the number of sessions or timeouts.

Exception Code	Description	Notes
QDocException016	Failed to parse the QDoc schema	While processing a QDoc in the service interface API adapter or updating the response QDoc when `suppressResponseDetail` is set to false for the UI API adapter, the respective request and response schemas are loaded into memory and parsed. If an error occurs parsing these schemas, this exception is returned. Review the schemas and the exception detail.
QDocException017	Failed to map the QDoc request into a QdocResultSet	See QDocException016.
QDocException018	Failed to construct a new IQDocMapper instance	`qxtend.jar` does not contain the required classes. Contact your system administrator.
QDocException019	Processing of request on Progress AppServer connection failed	This applies to the service interface API. An unexpected exception occurred during the processing. See exception detail and logs.
QDocException020	QDoc processing failed with an unhandled exception	This applies to the service interface API. An unexpected exception occurred during the setup or processing of the QDoc. View the response QDoc and log files for further information.
QDocException021	Error on startup of the QXtend server	QXtend startup failed to launch all of the managers, such as Configuration, Connection Pool, Queue, and so on. View `qdocInstall.log`.
QDocException022	Failed to map the QDoc schema into a sequence of fields used to create the response QDoc	Edit the QDoc so that the schema is mapped correctly.
QDocException023	Failed to create data in the QDoc response	This applies to the service interface API. Creating the response failed. View the response QDoc and log files for further information.
QDocException024	Reading data from a result set returned from the AppServer failed	Failed to extract the response returned from Progress, or failed to add error data returned from the service interface API adapter to the response QDoc. View exception details.
QDocException025	Request prevented from being passed to the API	No response data has been returned from the service interface API adapter. View the QDoc response.
QDocException026	Error during an API request	This exception returns Progress errors from processing QDocs through the service interface API adapter to the QDoc response.

Exception Code	Description	Notes
QDocException027	Failed to locate specified file	A schema or events file was not located for a QDoc. Review the response QDoc and ensure the files are in the correct location.
QDocException028	QAD and custom namespaces are identical	Custom QDocs should have a namespace URI that is appropriate to your organization. Do not use the same namespace as that used for standard QDocs. The default namespace for QAD QDocs is: `urn:schemas-qad-com:xml-services`. View the file `implementationQdocs.xml` in the relevant receiver directory to check the namespace.
QDocException029	Could not extract header from SOAP envelope	The SOAP envelope header does not exist or is in the incorrect format. See “QDoc Message Envelope” on page 301 for sample SOAP envelopes.
QDocException030	Could not extract body from SOAP envelope	The body of the SOAP envelope does not exist or is in the incorrect format.
QDocException031	Could not set header in SOAP envelope	The SOAP header was not imported into the response QDoc.
QDocException032	Could not set body in SOAP envelope	The SOAP body was not imported into the response QDoc.
QDocException033	Request file cannot be found in request queue	This applies to the Queue Manager. A request QDoc was removed from the directory queue during processing.
QDocException034	Error processing request	This applies to the Queue Manager. This is a serious technical exception caused by an installation or environment problem, and should never occur. Review `queue.log` and `qxtendserver.log`. Contact your system administrator and QAD Support.
QDocException035	Problem creating class	A problem was encountered when creating a class.
QDocException036	Error getting request	This applies to the Queue Manager. The Queue Manager polls the request directory for a list of files, then loads the files for processing. If a request QDoc is deleted between these events, this exception is returned. View the `queue.log`.
QDocException037	No request directory found in path	This applies to the Queue Manager. The path to the request directory is invalid. Check `environmentmanager.xml`. Also check that a `requests` directory exists in the relevant queue folder.

Exception Code	Description	Notes
QDocException039	Error writing response to file	This applies to the Queue Manager. QXtend was unable to write the response QDoc to the responses directory. This may be due to a permissions issue, or the directory was deleted or moved. Verify the response directory exists with correct permissions.
QDocException040	Request file is not in a working state	This applies to the Queue Manager. Moving a file that has not been processed to the responses directory failed. This occurs if duplicate file names are used.
QDocException041	Error writing contents of request file	This applies to the Queue Manager. QXtend is unable to write the request QDoc to the response directory. This is a permissions issue or the directory was deleted or moved. Verify the response directory exists with correct permissions. It is also possible that the response data is invalid. View `queue.log` and `qdocInfo.log`.
QDocException042	Error renaming file	This applies to the Queue Manager. QXtend is unable change the request QDoc to a processing state. This may be due to a permissions issue. Check permissions in the request directory. View `queue.log` and `qdocInfo.log`.
QDocException043	Unreachable destination, invalid request, missing SOAP envelope	This applies to the Queue Manager. The Web service is down or the link is incorrect. View the request and response QDocs, `queue.log`, and `qdocInfo.log`.
QDocException044	Problem wrapping request in envelope	This applies to the Queue Manager. Adding a SOAP envelope to the QDoc through the Queue Manager failed. This may be due to a permissions issue. Verify correct permissions on the request QDoc.
QDocException045	Error occurred while adding the envelope	This applies to the Queue Manager. See QDocException044.
QDocException046	Error occurred when parsing the response	This applies to the Queue Manager. Writing the response QDoc to the response directory failed. A default QDoc response is created and written to the responses directory error detail. A permissions problem with the queue directories is also possible. Check `queue.log` and `qdocInfo.log` for details.

Exception Code	Description	Notes
QDocException047	Queue Manager is in error	This applies to the Queue Manager. QXtend cannot access the Queue Manager. View `queue.log`, `qdocInfo.log`, and `qdocInstall.log` for details.
QDocException048	Error launching Queue Manager	This applies to the Queue Manager. An exception was raised during the startup of the Queue Manager and is logged to `queue.log` and `qdocInstall.log`. The Queue UI link displays in red.
QDocException049	Could not find response file	This applies to the Queue Manager. Queue Manager is unable to write the response document to the response directory. Verify that the directory exists. View `queue.log` and `qdocInfo.log`.
QDocException050	Could not create response document from QXtend output	This applies to the Queue Manager. Retrieving response details from QXtend failed. A default QDoc response is created and written to the response directory with error details. View `queue.log` and `qdocInfo.log`.
QDocException051	Could not find request file	See QDocException036.
QDocException052	Could not write XML to file	See QDocException039.
QDocException053	Could not create directory	QXI is unable to create a directory during the configuration of a queue or receiver. This is possibly a permissions problem. Verify permissions on the corresponding directories.
QDocException057	Could not read request file	The request QDoc has incorrect format or contains invalid characters.
QDocException058	Could not write to request file	See QDocException041.
QDocException059	Import or include schema has invalid namespace	A node in the QDoc schema does not have an associated namespace.
QDocException060	Import or include schema has invalid schema location	An import schema in the QDoc is invalid. Ensure the QDoc notation is correct.
QDocException061	Request requires authentication details	If this error is raised, the QDoc contains blank or missing username and password or session ID entries, but the receiver requires authentication information in the QDoc.
QDocException063	Unable to get an instance of the appropriate connection pool	The connection pool has not started. Check the log for errors.

Exception Code	Description	Notes
QDocException064	Invalid QDoc schema for XML syntax 1.1	Check that the correct syntax is being used for the schema in use.
QDocException065	Failed to map the QDoc schema into a ProDataGraphMetaData	Check the QDoc and modify as appropriate to correctly map the schema.
QDocException066	Failed to map the QDoc request into a ProDataGraph	Check the QDoc and modify as appropriate to correct the mapping issue.
QDocException067	Failed reading data from a ProDataGraph returned from the Progress AppServer	Failed reading data from a ProDataGraph returned from the Progress AppServer
QDocException068	Only supports QDoc 1.1	Check the syntax version in use.
QDocException069	$1 element in QDoc request does not match schema file	Check the syntax version in use.
QDocException070	Failed to convert ProDataGraph to XML	Check the QDoc and modify as appropriate to correctly map the schema.
QDocException071	Unable to create sample QDoc request	This option creates a sample QDoc based on an XML schema you specify. The schema must be one that is supported by QXI. The schema is parsed and all possible entries from the QDoc schema are created in the sample QDoc. The sample QDoc shows the way to represent all of the possible data structures in the QDoc. The generated QDoc shows the structure and content of the QDoc that needs to be sent to QXI. The generated schema also contains a valid SOAP envelope.
QDocException072	Unable to parse QDoc schema	Check the schema syntax in use.
QDocException073	AppServer error	Cannot pass data to the AppServer. Check the AppServer setting and the data to be sent.
QDocException074	Cannot get the QDoc XML syntax.	Check whether the schema has been added to the receiver. Also check that the XML syntax is correct.
QDocException075	The QDoc XML syntax is incorrect.	Check the XML syntax of the required QDoc in the API page.

Configuration Exceptions

These exceptions relate to errors which may occur while configuring QXI. These exceptions are displayed on the UI when configuring the system; details are available in `qxtendserver.log` and `qdocInstall.log`.

Table 28.10
Configuration Exceptions

Exception Code	Description	Notes
ConfigurationException001	Receiver specified is not valid	While adding a new schema, the receiver assigned does not exist.
ConfigurationException002	Receiver directory not present	While deleting a receiver, the receiver directory does not exist.
ConfigurationException003	Reloading managers failed	QXtend launches managers that carry out specific functions, for example, Queue Manager to allow QDoc processing through a queue and Configuration Manager to configure QXI. If QXI is restarted using the Tomcat admin functionality or the restart option in QXtend, these managers are reloaded. If some configuration has changed prior to reloading which invalidates any of the managers, then the QXtend UI link displays red and this exception appears in `qdocInstall.log` with more information.
ConfigurationException004	Schema descriptor info incorrect	When modifying a schema, the schema details are retrieved from a configuration file. If the details are unavailable, this error is raised. This occurs if the configuration file is directly modified and that record deleted. Review the configuration for this schema and reinstall if necessary.
ConfigurationException005	Events directory not found	While adding or updating a schema, the events directory does not exist.
ConfigurationException008	Events file not found	When adding or modifying a schema, the events file must exist in the specified location. If not, this exception is raised.
ConfigurationException009	Error occurred configuring the system	An exception occurred on QXtend startup related to the configuration of QXtend. The QXtend link displays red. The error is logged in `qdocInstall.log`.
ConfigurationException010	Error getting API configuration details	This error displays on the UI if an error occurs removing API support for a specified schema. This should only occur if the configuration files were manually updated. Contact the system administrator or QAD Support.
ConfigurationException011	Duplicate entries for an API	`QADQdocs.xml` or `implementationQdocs.xml` were incorrectly modified. View logs for details and contact the system administrator or QAD Support.

Exception Code	Description	Notes
ConfigurationException012	Referenced node missing in descriptor file	See ConfigurationException004.
ConfigurationException013	Error loading configuration details	An error occurred loading a configuration file. The file may have been manually moved or deleted, or manually updated with invalid data. View logs for details. Also verify the path and file permissions.
ConfigurationException014	Schema files do not exist in specified directory	While adding or deleting a schema, the schema files cannot be located. Verify path and permissions.
ConfigurationException016	File already exists	While creating or modifying a schema, the descriptor file already exists. Verify path and permissions.
ConfigurationException017	Problem writing to new file	While adding a new schema, the schema files already exist.
ConfigurationException018	Duplicate nodes exist in descriptor file	You cannot create an entry in a descriptor file for a node that already exists.
ConfigurationException019	Schema directory not found	While adding a schema, the schema directory does not exist or is read-only. Verify path and permissions.
ConfigurationException020	Deleting receiver directory failed	While deleting a receiver, the write to the backup directory failed.
ConfigurationException021	Receiver name cannot be null	While adding a receiver, the receiver name is blank or null. The null state may be raised if an error occurs recovering the value from the UI. Refresh and restart.
ConfigurationException022	Version not installed	While adding a receiver, the schema directory does not exist, could not be created, or the receiver version is blank.
ConfigurationException024	Receiver already supported	You cannot add a new receiver that already exists for that version. Specify a different receiver or edit existing receiver.
ConfigurationException025	Receiver details not found	While deleting a receiver, cannot find the receiver node in `receivers.xml`.
ConfigurationException026	Parsing descriptorInfo failed	Building a hashmap of nodes from a descriptor file failed. Verify that the file exists.
ConfigurationException027	Updating descriptorInfo failed	This error occurs while creating a new receiver structure, path directory, or subdirectory for requests. It can also occur while updating global or receiver descriptor files with a new node.

Exception Code	Description	Notes
ConfigurationException028	Specified configuration file not found	The configuration files `qxtendalertconfig.xml`, `qxtendalertgroupconfig.xml`, and `qxtendalertuserconfig.xml` are missing from `WEB-INF/emailAlerts`. Replace the files.
ConfigurationException029	Alert recipient must be assigned a value	The Alert Recipient field is blank. Enter a valid e-mail address.
ConfigurationException030	Alert recipient's e-mail address is invalid	The specified e-mail address uses an incorrect format. The correct format is `username@domain`.
ConfigurationException031	Alert recipient already exists	The alert recipient entered already exists. Enter a different recipient.
ConfigurationException032	A problem occurred when updating the configuration file	The configuration files `qxtendalertconfig.xml`, `qxtendalertgroupconfig.xml`, and `qxtendalertuserconfig.xml` cannot be updated. Check that the `WEB-INF/emailAlerts` directory exists and file permissions.
ConfigurationException033	Cannot map schema and XML files to ProDataGraph	Check the log file.
ConfigurationException035	Cannot create menu XML	Check the log file.
ConfigurationException036	Problem occurred when reading XML	The configuration files `qxtendalertconfig.xml`, `qxtendalertgroupconfig.xml`, and `qxtendalertuserconfig.xml` are not well-formed or contain incorrect data. Check the files.
ConfigurationException037	Alert group must be assigned a value	The Alert Group field is blank. Enter an alert group name.
ConfigurationException038	Alert group already exists	There is already an alert group with the alert group name just entered. Enter a different name.
ConfigurationException039	Cannot find alert receiver	The alert receiver specified may have been deleted or modified by another user. Refresh the screen to get the latest information.

Exception Code	Description	Notes
ConfigurationException040	Domain name must be assigned a value	The Alert Domain field is empty. Enter a domain name.
ConfigurationException041	Domain name already exists	There is already a domain using the domain name just entered. Enter a different name.
ConfigurationException042	Problem occurred removing receiver in alert	The receiver may have been deleted or modified by another user. Refresh the screen to get the latest information.
ConfigurationException043	Problem occurred creating a session to send mail	There was a JavaMail error.
ConfigurationException044	Problem occurred creating a mime to send mail	There was a JavaMail error.
ConfigurationException045	Problem occurred setting mail subject	The e-mail subject is incorrect. If the alert group is using the default message configuration, check the `qxtendalertconfig.xml` file in the `WEB-INF/emailAlert` directory. If it is not using the default message configuration, check the e-mail subject setting in the Group Configuration Parameters page.
ConfigurationException046	Problem occurred setting mail body	The e-mail body is incorrect. If the alert group is using the default message configuration, check the `qxtendalertconfig.xml` file in the `WEB-INF/emailAlert` directory. If it is not using the default message configuration, check the e-mail body setting in the Group Configuration Parameters page.
ConfigurationException047	Problem occurred adding mail attachment	Cannot add the attachment. Check the log for more details.
ConfigurationException048	Problem occurred setting the e-mail “From:” in the mail header	The e-mail “From:” is incorrect. Check the setting in the Email Configuration Parameters screen.
ConfigurationException049	Problem occurred setting the e-mail “To:” in the mail header	The e-mail “To:” is incorrect. Check the setting in the Email Configuration Parameters screen.

Exception Code	Description	Notes
ConfigurationException050	Problem occurred sending mail	An error occurred when sending the e-mail. Check the SMTP setting, e-mail body, e-mail subject, and attachments.
ConfigurationException051	SMTP server must be assigned a value	The SMTP Server field is blank. Enter the name of an SMTP server.
ConfigurationException052	The “From:” e-mail address is invalid	The specified “From:” e-mail address is incorrect. The correct e-mail address format is `username@domain`.
ConfigurationException053	Cannot find alert domain	The specified alert domain may have been deleted or modified by another user. Refresh the screen to get the latest information.
ConfigurationException055	SMTP settings are invalid	The SMTP settings are incorrect. Check the SMTP server, port number, and authentication details in the Email Configuration Parameters screen.
ConfigurationException056	Recipient must be assigned a value	The Alert Recipient field is blank. Enter a valid e-mail address.
ConfigurationException057	Recipient’s e-mail address is invalid	The format of the e-mail address entered for the recipient is incorrect. The correct format is `username@domain`.
ConfigurationException058	No recipients	There are no recipients specified for the e-mail. Check the alert recipients setting in the Group Configuration Parameters screen.
ConfigurationException059	The “From:” must be assigned a value	The Email From field is blank. Enter a valid e-mail address.

Process Exceptions

These exceptions relate to errors which may occur during the processing of a QDoc. These exceptions appear in the response QDoc and/or in log files.

Table 28.11
Process Exceptions

Exception Code	Description	Notes
ProcessException001	Session failed to initialize	QDoc processing program failed to launch. Review the Connection Pool Manager settings for the pool and test the initialization by manually following the pool log-in script.
ProcessException002	Failure to send message to begin submitting data	Data submission to the server failed. An acknowledgement may fail due to the trigger failing to fire. To find the root cause, follow the process through a character session and look for any unusual user interface functions such as alert boxes, selection lists, or browses.
ProcessException003	Failure to send action or data message	See ProcessException002.
ProcessException005	Failure to send spacebar event	QXtend is unable to respond to a pause event that occurs in QAD Enterprise Applications. Enable QXtend logging to the warn level and follow the process through a character session, looking for any unusual interface functions around that field. See “Log Report Levels” on page 158.
ProcessException007	Java encoding specification sent from Progress failed	The QAD Enterprise Applications session request for user encoding from QXtend returned an invalid encoding string. Check the user encoding. This exception may also occur if the session failed to initialize correctly. Check the initialization procedure described in ProcessException001.
ProcessException009	Timeout occurs waiting for response from QAD Enterprise Applications QdocReceiver Configuration Manager	Communication occurs constantly between QXtend and QAD Enterprise Applications, from session start to the processing of the QDoc. QXI defines a timeout for a response to take place from QAD Enterprise Applications. If the expected response is not received within this timeout period, this exception is returned. May occur if a record is locked or if the terminal was exited abnormally.
ProcessException010	Error while attempting to reset a Progress AppServer connection	Closing the connection to the application server raises an error. Check the details on the exception for details.

Failure Exceptions

These are low-level exceptions, which, in most cases, are wrapped within another exception and are only seen at the lower level of a stack trace. These exceptions appear in the response QDoc and/or in log files.

Table 28.12
Failure Exceptions

Exception Code	Description	Notes
FailureException003	Writing document to file failed	While updating configuration settings using the Connection Pool Manager, the new settings cannot be written to a file. For details about avoiding FailureException responses in HP-UX, COMPAQ, or LINUX environments, see “FailureException or AdapterException Responses” on page 322.
FailureException005	Loading environmentmanager manager classes failed	QXtend startup failed to launch all of the managers, such as Configuration, Connection Pool, Queue, and so on. View `qdocInstall.log`.
FailureException006	New XML document instance create failed	This error occurs if QXtend attempts to read in an invalid XML document.
FailureException007	Unknown error occurred instantiating manager class	An error occurs starting up QXtend. This may be caused by invalid configuration files. View `qdocInstall.log`.
FailureException008	Unable to find or load configuration file for manager class	A configuration file was not found during QXtend startup. View `qdocInstall.log`.
FailureException009	Unable to find or load configuration file	A configuration file listed in `config-files.xml` was not found during QXtend startup. View `qdocInstall.log`.
FailureException010	Timeout on sending data	The QAD Enterprise Applications session may be locked out by another user.
FailureException011	Invalid locale information returned from Progress	There is a Progress error on the QAD Enterprise Applications client.
FailureException012	Invalid system information returned from Progress	The QXtend client files on QAD Enterprise Applications may be the incorrect version.

Internationalization Exceptions

These exceptions may arise if, for example, multiple languages are used and there is no resource file for the language in use.

Table 28.13
Internationalization Exceptions

Exception Code	Description	Notes
I18NException001	Error processing - invalid locale	An invalid locale was encountered during processing.
I18NExceptionex002	Missing resource bundle	A resource bundle is missing.
I18NExceptionex003	Locale does not exist	The specified locale does not exit. Check the locale specified.

Reflection Exceptions

These exceptions may arise due to internal faults with the engine itself—for example, missing Java classes arising from a faulty installation.

Table 28.14
Reflection Exceptions

Exception Code	Description	Notes
ReflectionException001	Error while creating instance of class	An error was encountered when creating an instance of a class.
ReflectionExceptio002	Error invoking given method	An error was encountered when invoking a given method.

License Exceptions

These exceptions may arise if agents are unavailable or if the license code is invalid.

Table 28.15
License Exceptions

Exception Code	Description	Notes
LicenseExceptionex008	The license code is corrupt.	The license code is corrupt or is invalid. Retype the license code. If the problem persists, contact QAD support.
LicenseExceptionex009	The license code is blank.	A blank license code has been submitted.
LicenseExceptionex015	No Licensed Agents available.	There are no available licensed agents for the request. If the problem occurs frequently, extra agents may need to be purchased.

Chapter 29

QXO Entity Relationships and Tables Reference

This section contains QAD QXtend Outbound (QXO) entity relationship diagrams and table descriptions for the QXtend database.

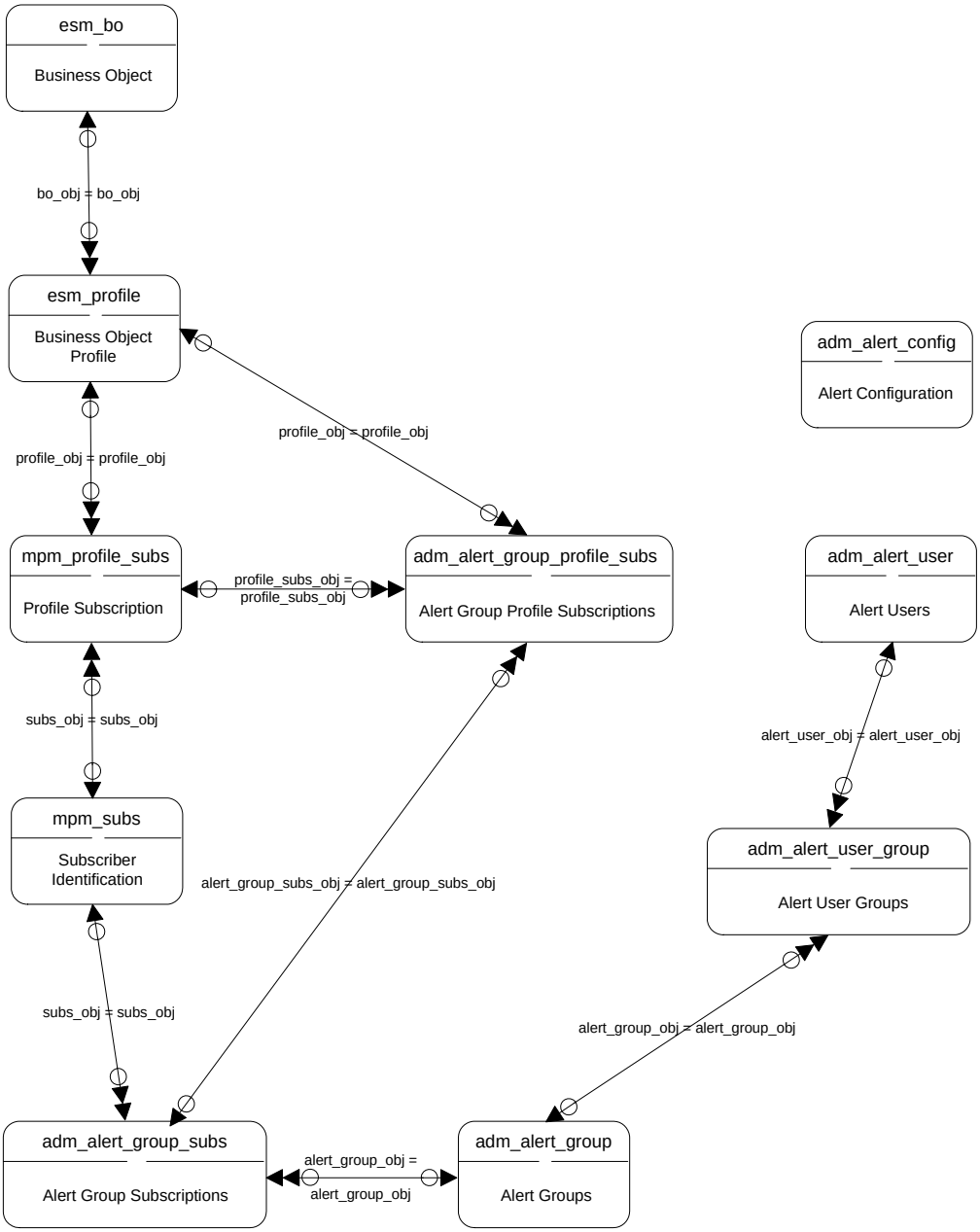
Entity Diagrams

This section includes diagrams that detail key relationships among various tables in the QXtend database.

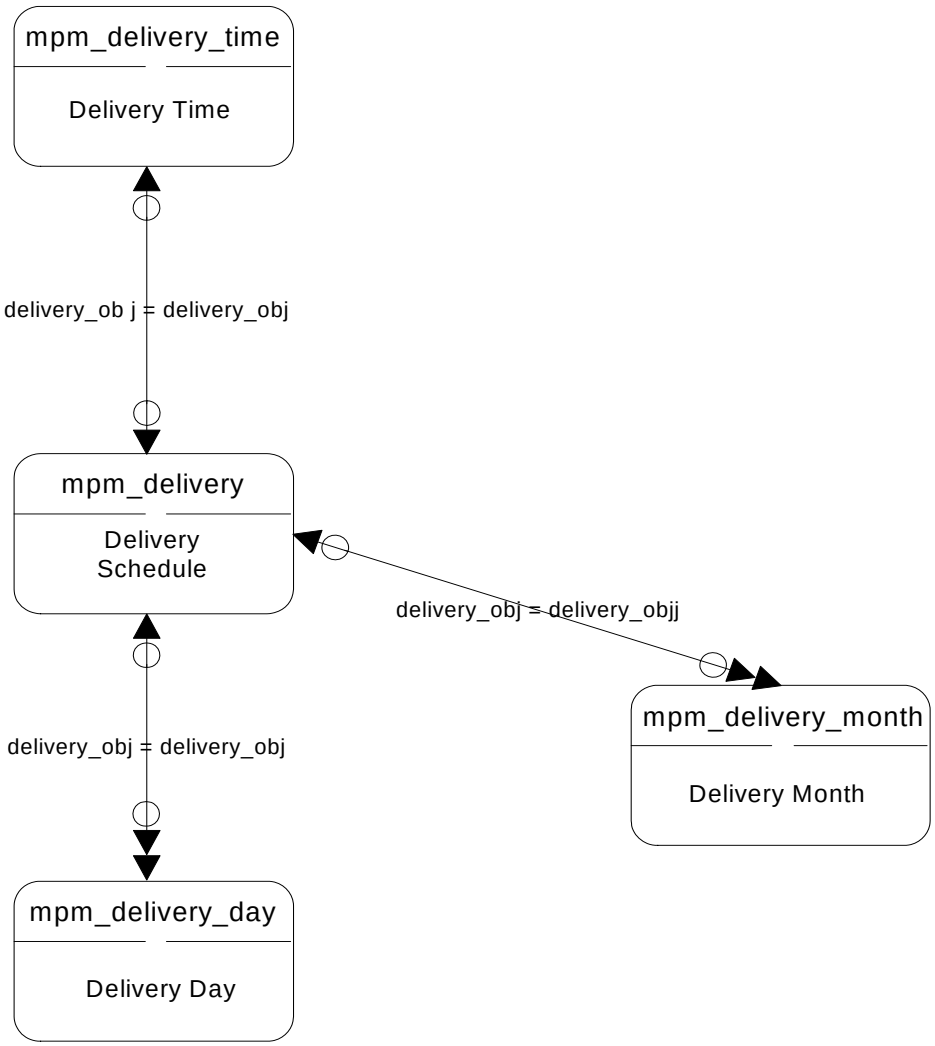
Note For details about conventions used in entity relationship diagrams (ERDs), refer to *Entity Diagrams*.

These diagrams should be used in conjunction with the section “QXtend Table Descriptions” on page 383. The entity diagrams are in alphabetical order starting on the next page.

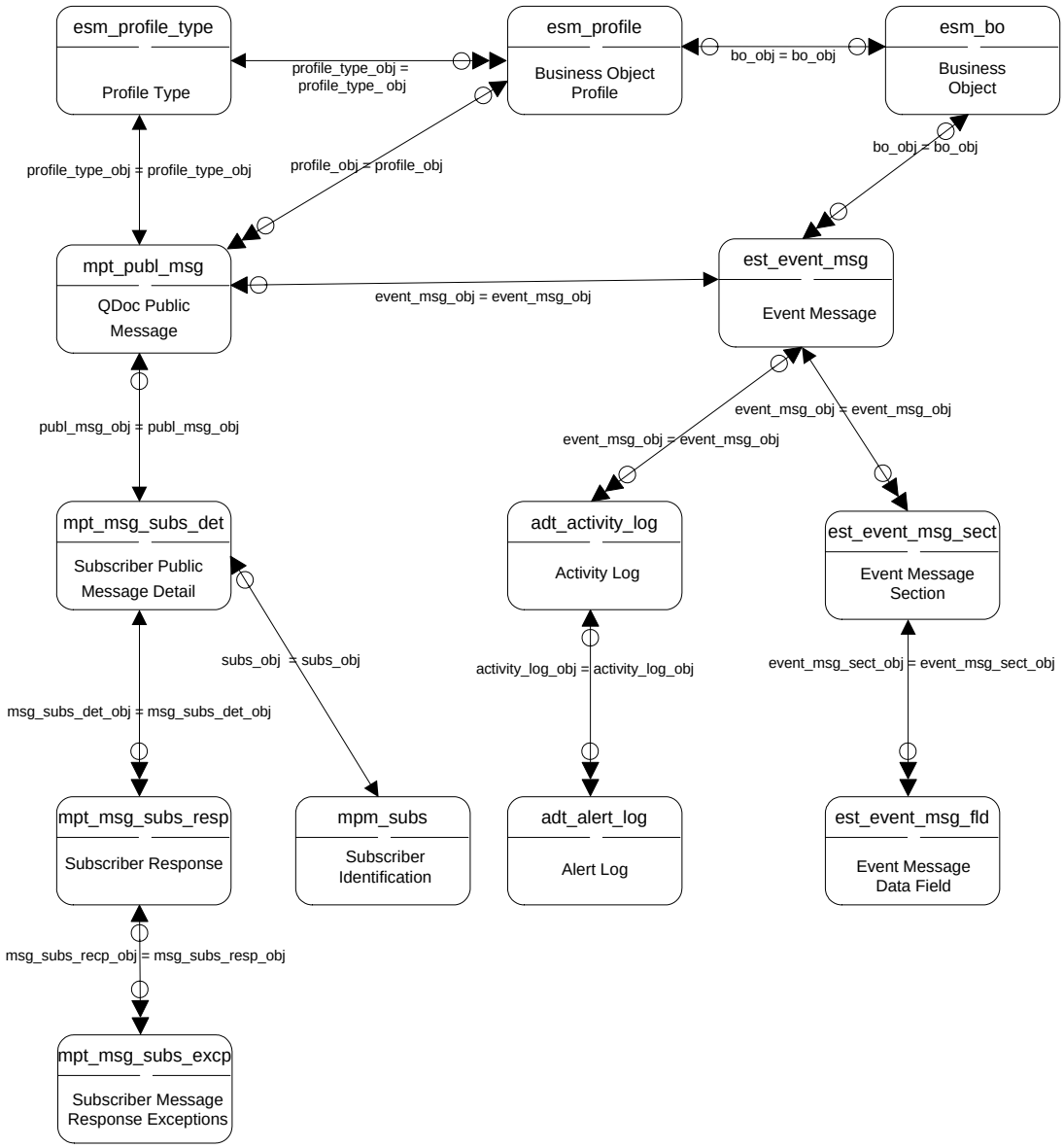
E-mail Alerts



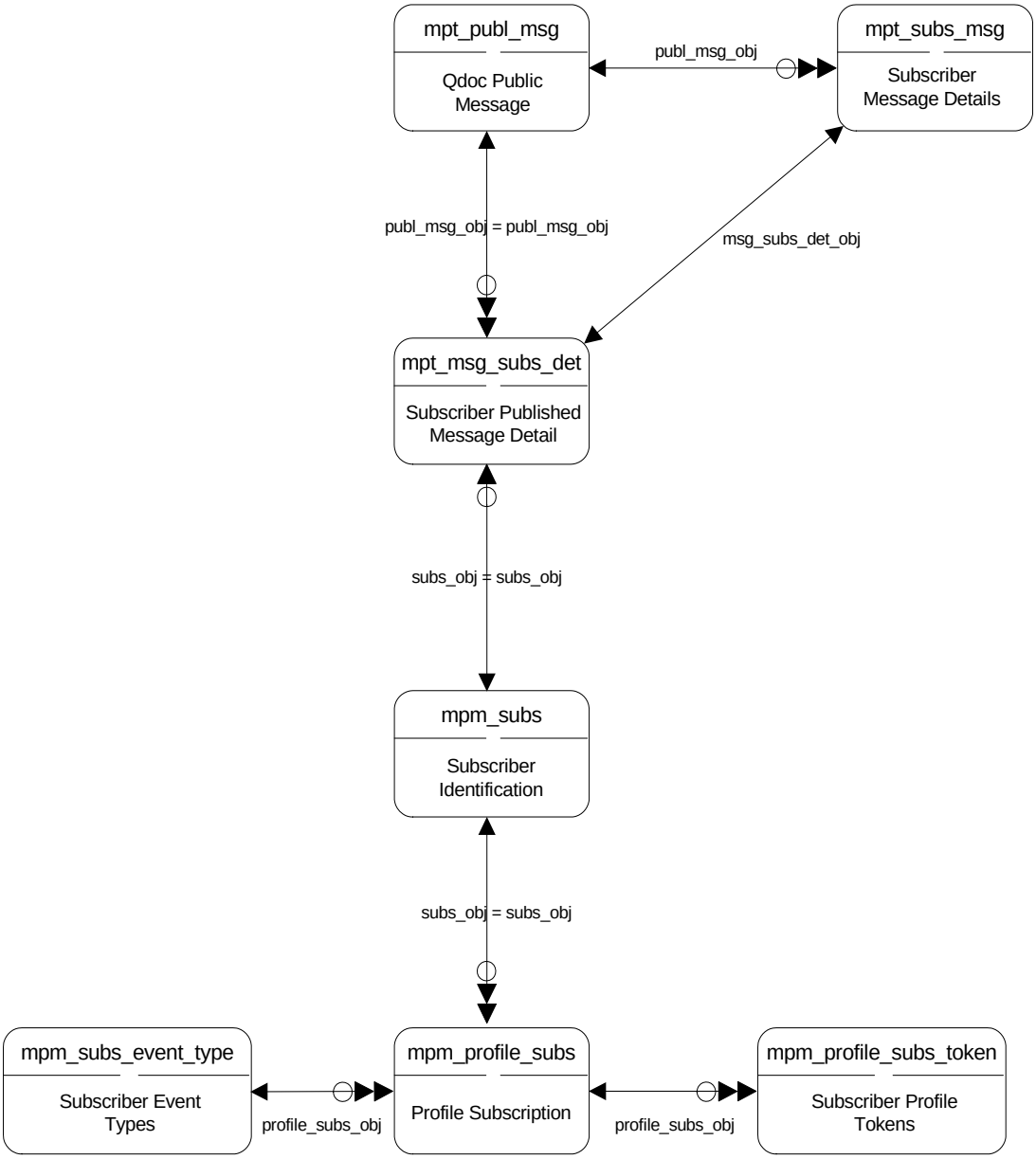
Delivery Scheduler



Message Traceability



Subscribers



QXtend Table Descriptions

This section includes an alphabetical listing of tables used in the QXtend schema.

These definitions are meant to be used in conjunction with the section “Entity Diagrams” on page 378.

Table	Name	Description
adm_alert_config	Alert Configuration	This entity specifies general configuration for the alerts functionality. It defines the generic parameters required to send an e-mail alert.
adm_alert_group	Alert Group	This entity defines a collection of users and configuration parameters specific to that group of users. It is used to assign parameters defining the appropriateness of receiving an alert.
adm_alert_group_profile_subs	Alert Group Profile Subscriptions	This entity defines registrations between a group and profile such that the group is assigned to receive alerts for that profile.
adm_alert_group_subs	Alert Group Subscriptions	This entity defines registrations between a group and subscriber such that the group is assigned to receive alerts for that subscriber.
adm_alert_metric	Alert Metric	This entity defines a metric that is checked at runtime for all sessions derived from the session profile for which alerts are raised when stated conditions are not met. The metrics must be members of the set supported by QXtend Outbound.
adm_alert_user	Alert User	This entity defines an individual user (email address) that enables it to be registered to groups to receive alerts.
adm_alert_user_group	Alert User Group	This entity defines registrations between a user and group such that all members of a group will receive an alert if required.
adm_event_type	Alert Event Type	This entity defines the event types for the source application type.
adm_event_type_src_app	Alert Event Type	This entity defines the event types for the source application.
adm_evt_sess_link	Event Type Session Link	This entity maintains event type registration of a registered source application of an event service.

Table	Name	Description
adm_session_profile	Session Profile	This entity defines and configures an event service, associates each one with one or more event recorders and an AppServer, and determines how event service instances are allocated to the source applications at runtime.
adm_sessprof_srcapp	Session Profile Source Applications	This entity lists source application codes that are serviced by profile sessions. It is populated for event service session profiles only.
adm_src_app	Source Application	This entity defines a source application whose events are published by QXtend Outbound.
adm_src_apptype	Application Type	This entity includes application type codes that represents source applications with the same structure. These applications may share the same business object definitions because their schema is the same.
adm_system_code	Outbound Parameter Settings	This table stores the various QXtend Outbound parameter settings in a hierarchical structure similar to that of an XML document.
adt_activity_log	Activity Log	This entity provides an audit trail of each significant QXtend Outbound activity, in the form of a comprehensive log record suitable for export to spreadsheet applications or other analysis/reporting tools. Activities recorded include session startup and shutdown, application event processing, and event message processing. The record references any associated application event or event message and provides the raw detail used to calculate overall QXtend Outbound operational statistics.
adt_alert_log	Alert Log	This entity stores alert conditions raised by QXtend Outbound processes and sends them to the attention of a system administrator. Where available, it references the activity log record for each alert condition.
adt_email_queue	Alert Email Queue	This entity contains the queue of email alerts ready to be delivered.
adt_msg_params	Alert Message Parameters	This entity contains the message parameters passed between the Outbound services.
adt_session	Session	This entity describes the state of each running event service, message publisher, message sender, or subscriber services session, providing current processing status and basic throughput statistics.
adt_session_alive	Alive Sessions	This entity indicates whether a service is running.

Table	Name	Description
adt_src_app_db	Source Application Databases	This entity stores the database connection information for one database that is part of a source application.
adt_summary	Alert Summary	This entity contains the subscriber message summary.
adt_svc_alerts	Source Alert Conditions	This entity stores alert conditions raised by QXtend Outbound processes to the attention of a system administrator. It references the source process of each alert condition.
esm_bo	Business Object	This entity defines the business object. Business objects specify all application data available for publishing, constraining the definitions of subscription profiles. They query the application databases and extract the data to be published for each event.
esm_bo_do	Business Object Data Table	This entity defines a table contained in the business object that lists the joining relationship with the business object parent table and other extraction-related attributes. It references a child entity in the QXtend Outbound database to maintain field-level detail for each table.
esm_bo_event_type	Business Object Event Types	This entity contains the various event types for the business objects.
esm_bo_fld	Business Object Field	This entity identifies a field from a database table included in the business object or subscription profile and specifies data-formatting details inside outbound messages. Additional derived fields to be published are also defined. It is not maintained for all fields, only those using any attribute to control publishing. Note: This table is obsolete as of version 1.4.
esm_bo_group	Business Object Group	This entity defines groups of business objects whose events must be published in chronological sequence. It is used to preserve the sequence of published outbound messages when all active event types within the group are all registered with the same event service.
esm_bo_group_ref	Event Group-Member Reference	This entity references each event group to its member event types.
esm_bo_sess_link	Business Object Session Link	This entity is an association table between esm_bo_profile and adm_session_profile.

Table	Name	Description
esm_msgprof_link	Message Profile Link	This entity maintains a link between a profile data table and the event message data table for determining if the record still exists in the source application.
esm_profile	Business Object Profile	This entity defines the business object. Business objects specify all application data available for publishing, constraining the definitions of subscription profiles. They also query the application databases and extract the data to be published for each event.
esm_profile_do	Business Object Profile Data	This entity defines a table contained in the business object that lists joining relationships with the business object parent table and other extraction-related attributes. It references a child entity in the QXtend Outbound database to maintain field-level detail for each table.
esm_profile_event_type	Profile Event Types	This entity contains the various event types for profiles.
esm_profile_fld	Business Object Profile Field	This entity identifies a field from a database table included in the business object or subscription profile and specifies data-formatting details inside outbound messages. Additional derived fields to be published are also defined. It is not maintained for all fields, only those using any attribute to control publishing.
esm_profile_type	Profile Type	This entity lists valid profile types that can be assigned to the esm_profile table.
est_event_msg	Event Message	This entity defines an event message consisting of a queued application event and the data regarding its associated business objects at the time the event was recorded. It is created and populated for each application event, and processed to create outbound messages for external subscribers.
est_event_msg_fld	Event Message Data Field	This entity defines a single data field and its value within a section of an event message. It represents the application data about the event that is available for publishing to external subscribers. This table is obsolete from version 1.4 onwards.

Table	Name	Description
est_event_msg_sect	Event Message Section	This entity defines the section of an event message containing the data extracted from one entity instance within the event's associated business object. Normally, each section contains the data from a single record in the application database.
est_extract_event	Extract Event	This entity records the events that identify the data to be extracted from the source application.
est_extract_row	Extract Row	This entity includes the row identifier of the top level record of a business object in the source application.
mpm_archive	Archive Settings	This entity identifies the settings for the archive service.
mpm_delivery	Delivery Schedule	This entity identifies a delivery schedule that can be attached to a subscriber or a subscriber profile.
mpm_delivery_day	Delivery Day	This entity identifies a day to send a document for a delivery schedule.
mpm_delivery_month	Delivery Monty	This entity identifies a sending month for a delivery schedule.
mpm_delivery_time	Delivery Time	This entity identifies a sending time for a delivery schedule.
mpm_profile_subs	Profile Subscription	This entity references a subscription profile to which an external system or application subscribes.
mpm_profile_subs_token	Subscriber Profile Tokens	This entity contains the tokens that can be used with subscriber profiles.
mpm_sesprof_subs	Message Sender Subscribers	This entity identifies the message sender process for a subscriber.
mpm_subs	Subscriber Identification	This entity identifies an external system or application that subscribes to one or more business objects and describes how they communicate with QXtend Outbound.
mpm_subs_comm_param	Subscriber Communication Parameters	This entity defines the communications-related settings used to send messages to the external subscriber. The settings must be from the predefined set supported by QXtend Outbound for the subscriber’s communications method.
mpm_subs_event_type	Subscriber Event Types	This entity defines the event types for subscribers.
mpm_subs_srcapp	Subscriber Source Applications	This entity lists source applications that the subscriber is interested in.

Table	Name	Description
mpm_subs_type	Subscriber Type	This entity identifies a subscriber type with which any external subscriber must be associated. The contents are populated at QAD and distributed to the customer, not maintained by users. All fields are encrypted before being stored to prevent users from making unauthorized changes to their QXtend Outbound license.
mpt_msg_subs_det	Subscriber Published Message Detail	This entity references a business object and external subscriber to which the outbound message is to be published. It is used to track the delivery of each message to all its subscribers.
mpt_msg_subs_excp	Subscriber Message Response Exceptions	This entity lists exceptions contained in a specific subscriber response message.
mpt_msg_subs_lock	Message Subscriber Row ID	This entity indicates that a subscriber message for a particular rowid in the database is being sent. Only one message per rowid per subscriber can be sent at any one time.
mpt_msg_subs_resp	Subscriber Response	This entity contains the response information for a specific attempt to send the QDoc to the subscriber.
mpt_publ_msg	QDoc Public Message	This entity defines a QDoc XML message created from an event message and to be published to one or more subscribers. It stores the processing state of the message.
mpt_subs_msg	Subscriber Message Details	This entity contains the message details for subscribers.
qxodb_ctrl	Outbound Database Control	Database Control for QXODB.

Section 5

Appendix

This section includes the following appendixes:

Appendix A

Data Synchronization

This section describes how to synchronize data between QAD Enterprise Applications source and destination domains. It is intended for users of the QAD DataSync application.

Introduction

This document is designed to explain to users of QAD QXtend—comprising QAD QXtend Inbound (QXI) and QAD QXtend Outbound (QXO)—how to synchronize data between QAD Enterprise Applications source and destination domains.

Note The tasks described in this section are described in detail elsewhere in this user guide. This appendix is intended to provide an overview of these activities and how they support data synchronization. Use the cross-references provided to see details about performing these activities.

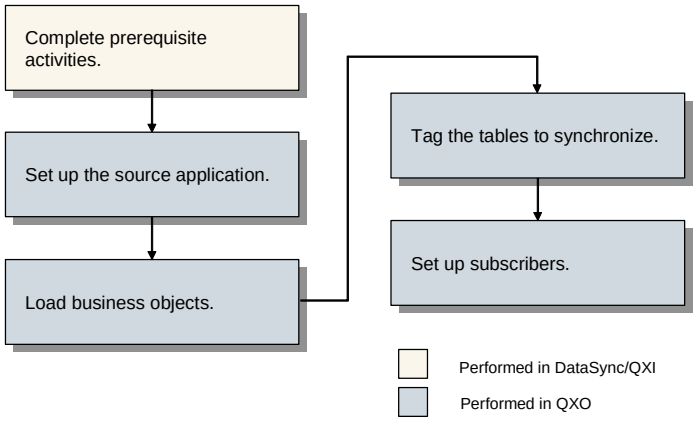
If you currently use QAD DataSync as your data synchronization solution, you can continue to do so. If you do not currently use DataSync—or if you intend to replace DataSync with the QAD QXtend—use this appendix as a guide to synchronizing data between domains in distributed QAD Enterprise Applications databases.

Important Only QDoc version 1.1 has been validated for QAD SE and QAD EE.

Data Synchronization Work Flow

Figure A.1 illustrates the data synchronization work flow; use it to set up data synchronization in your environment.

Fig. A.1
Data Synchronization Workflow



Before you can synchronize your data, several prerequisite steps must be completed—documenting your existing DataSync configuration, identifying your client domains, and so on.

Completing Prerequisite Activities

Before synchronizing your data, you must:

- Determine the data to be synchronized
- Define your receivers

Determining the Data to Synchronize

You can determine the data that needs synchronizing in either of two ways:

- If you currently use DataSync to synchronize data, by noting the setup of your synchronization profiles
- If you do not use DataSync, by performing an audit of your data synchronization requirements

Documenting Your DataSync Configuration

DataSync synchronizes static data among multiple, distributed QAD Enterprise Applications databases. Synchronization profiles specify which data—tables and fields—to synchronize between databases and which types of data changes—add, change, delete—to synchronize.

To identify the tables—and within the tables, the fields—that are currently synchronized, use Synchronization Profile Maintenance (36.8.22.1) to review your existing DataSync profiles. (Typically the name of the profile matches the name of the associated table.) The screen displays the fields in the table for which data will be sent.

You also can use Synchronization Profile Inquiry (36.8.22.2) to review the contents of your synchronization profiles. The inquiry shows the table the profile links to and the fields to be exported in that table. You must run an inquiry for each synchronization profile.

Auditing Your Data Synchronization Requirements

A complete description of how to perform an audit of your data synchronization requirements is beyond the scope of this document. However, your audit should answer the following types of questions:

- What data is shared by the QAD Enterprise Applications domains in my various sites or companies?
- Where does the maintenance of this data need to be centralized?
- Which domains will share the data?

After defining the types of data to be shared, relate the data back to the tables which control this data, and then identify the specific fields you require.

Refer to the list of business objects and associated tables that are provided with QXtend to see if the tables you have identified are included with this release. See “Business Objects and Outbound QDocs” on page 4.

If a suitable business object is provided, you may have to edit the business object to suit your specific purposes. If the tables you want to synchronize are not supported by a business object, you will have to create a new business object as well as develop a corresponding QDoc.

Define the Receivers

After planning which data to synchronize, the first implementation step in QXtend Inbound is to define your receiver or receivers. Receivers are named QAD Enterprise Applications instances. You need to define a receiver for each QAD Enterprise Applications database. Each receiver can supply data to multiple domains in the database.

Use the Configuration Manager in QXI to:

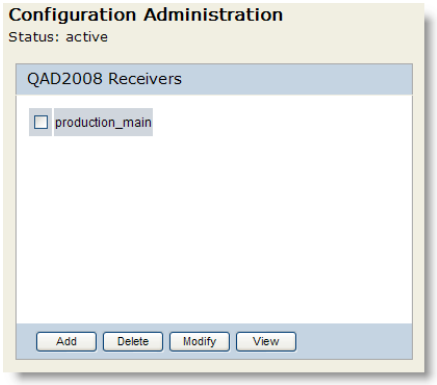
- Create a receiver for each destination database to receive data
- Assign QDocs to each receiver

For details about these steps, see “Inbound Receivers” on page 180.

Create a Receiver for Your Domains

Use the Configuration Administration screen (Configuration|Receivers) to create a new QAD2008 receiver for the destination database.

Fig. A.2
Add a Receiver



Click Add to add a new receiver. The name of the receiver should match the name of the QXI Connection Pool name. If you are running QXI you are prompted to suspend the application while adding the receiver.

Note For details about connection pools, see Chapter 17, “QXI Connection Pool Manager,” on page 201.

Choose Next to display the list of available QDocs.

Assign QDocs to the Receivers

In the Configuration Administration screen indicate which QDocs you want to assign to the receiver. The QDocs that display are part of the standard set that is supplied with QXI. Click Done when you have finished assigning the QDocs. If you suspended QXI you should restart it.

Setting Up Data Synchronization

After completing the prerequisite activities, you are ready to set up data synchronization in QXtend Outbound. You must:

- Set up the source applications and databases
- Load the business objects
- Tag the tables you want to synchronize
- Set up your subscribers

Example Scenario

A business object named Forecast is supplied with QXtend. Follow the descriptions in this section using the Forecast business object to understand how data is synchronized between the source domain (demo1) and destination domain (demo2).

The Forecast business object has been assigned the source application type QADSE. The source application type contains the source application named New York.

The Forecast business object has two profiles: PF - Forecast and PF - Forecast DataSync. The PF - Forecast profile is associated with the data object named Forecast with the table name fcs_sum (Forecast Summary). The appropriate event types for the application have been activated.

Set Up the Source Application

The next step is to set up your source application:

- Define and activate your source application type
- Define your source applications

Define and Activate Your Source Application Type

Source application types are the specific databases from which the QXO event service extracts data. Defining a source application type enables you to identify the event types that you want to extract from the databases within that source application type. Activating the source application type allows the system to extract the data.

Source applications are defined by using the Source Application Types screen (QXO Console|Configuration|Source Applications).

Note The QADSE source application type is delivered with QAD QXtend, but you must select the Active check box to activate it.

For details see “QXO Source Applications” on page 23.

Define Your Source Applications

Next, define the source applications from which you want to extract events. Source applications are the specific databases from which the QXO event service extracts data. Multiple source applications can belong to the same source application type as long as they use an identical database schema.

Use the Databases screen to define source application databases. Click Databases under the source application and add the new databases. You must define both the production `qaddb` database as well as the `qxevents` database.

For details see “QXO Source Applications” on page 23.

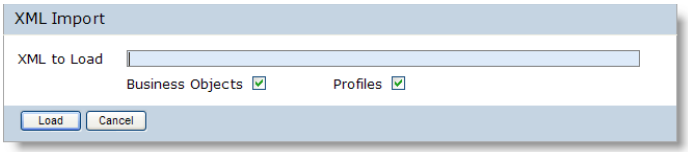
Import Your Business Objects

The next step is to import your business objects and profiles. You can import your business objects one at a time or all of them at the same time. Importing your business objects populates the `qxodb` database with data.

Note In order to be able to import your business objects and profiles, you must first have created your source application and databases. This procedure assumes that you can connect to those databases.

You use the XML Import screen (QXO Console|Configuration|Utilities|XML Import) to import your business objects.

Fig. A.3
Import Your Business Objects

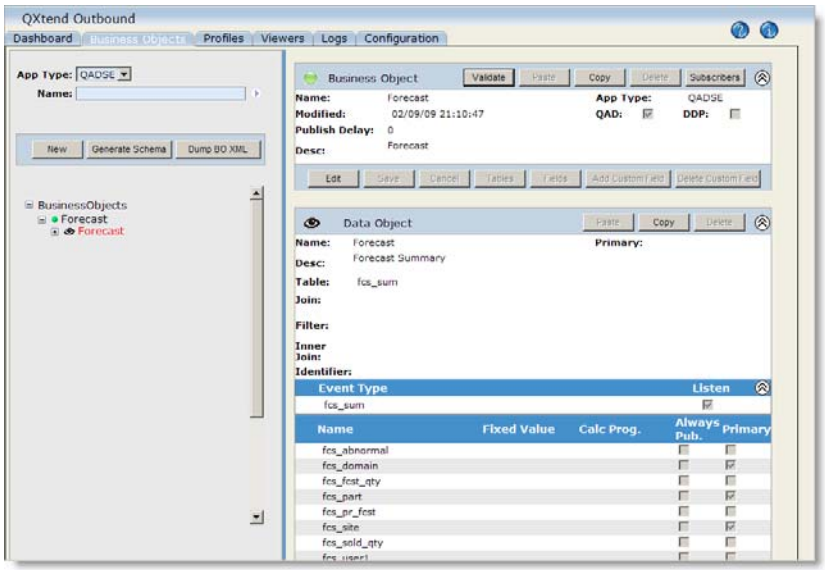


Our scenario uses the Forecast business object and a corresponding profile named MaintainForecast. To import business objects you type the location of your business object in the XML to Load field—for example, `QADSE/forecast/*`—select the Business Objects and Profiles check boxes, and then click Load.

For details see “Importing Business Objects and Profiles” on page 56.

You use the Business Object screen (QXO Console|Business Objects) to verify your business objects have been imported successfully into the database. Click the Forecast business object node to view the tables included in the business object and the associated data object.

Fig. A.4
Forecast Business Object



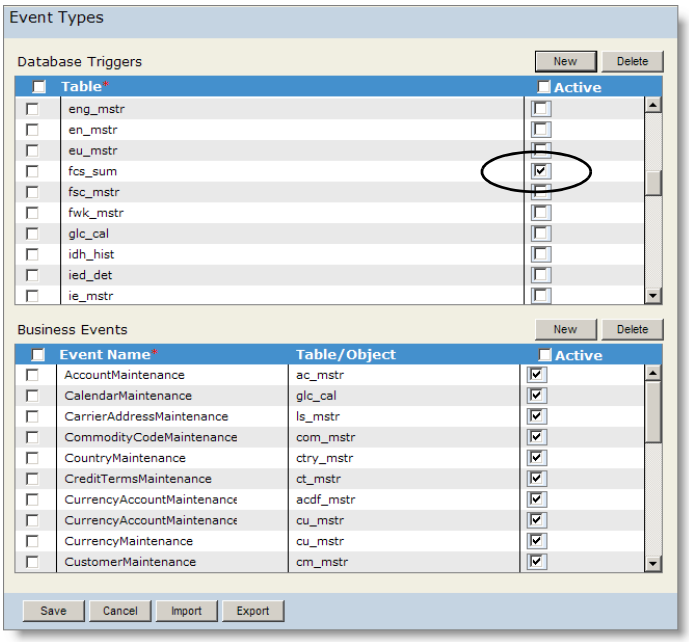
Activate the Tables to Synchronize

After importing your business objects and profiles, the next step is to activate the event types for the tables in the source application. You use the Event Types screen (Configuration|Source Applications|QADSE|*application*|Event Types) to activate your event types.

In our scenario the source application will trigger an event when the event service detects that the `fcs_sum` (Forecast Summary) table has been modified.

Important New customers to QXtend will always use QDoc Version 1.1. QDoc Version 1.0 is described for backward compatibility reasons.

Fig. A.5
Events Types

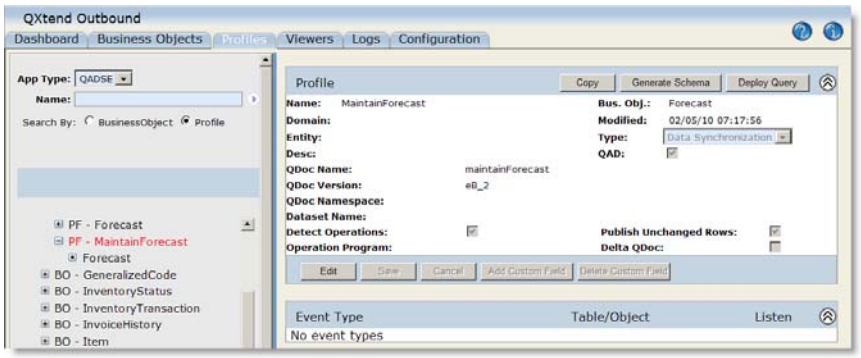


Note that in our scenario, in the Database Triggers section of the Event Types area, the check box in the Active column is selected for the `fcs_sum` table (indicated).

For details, see the section “Editing Source Application Event Types” on page 29.

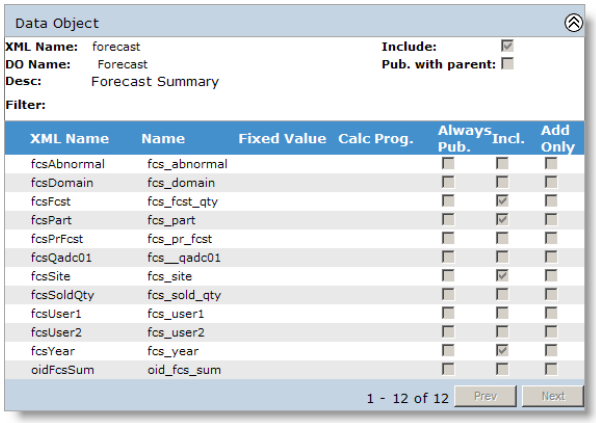
You can check the profiles associated with the Forecast business object. Click the Profiles tab and select the QADSE source application type. Click the Profiles option and perform the search. Expand the Profiles node and select the BO - Forecast business object. Select the PF - MaintainForecast.

Fig. A.6
DataSync Profiles



The profile takes the business object and converts it into a QXI QDoc. Click the Forecast data object node to view the Incl column which indicates the fields that will be included in the exported QDoc.

Fig. A.7
Forecast Data Object



Set Up Your Subscribers

The next step is to set up your subscribers, which involves two steps:

- Create a subscriber for each domain.
- Assign profiles to your subscribers.

Create a Subscriber for Each Domain

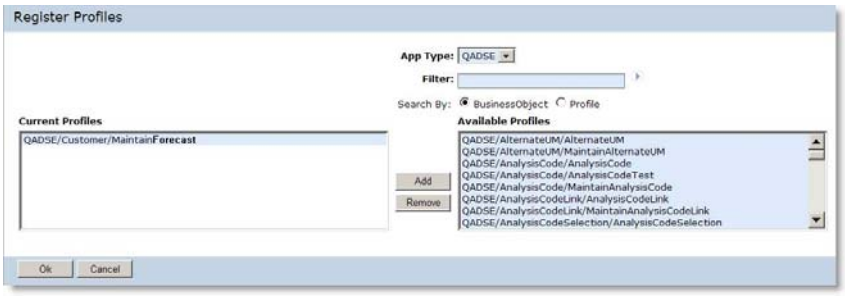
You configure subscribers using the Configuration Parameters screen (QXO Console|Configuration|Subscribers). In our scenario data is being synchronized between the source domain (demo1) and the destination domain (demo2).

In the Configuration Parameters screen, set the Communication Method to QXtend Web Service. Then specify the Tomcat location of QXI, the receiver, and domain. For details on defining a subscriber using the options for the QXtend Web Service method of communication, see “QXO Subscribers” on page 38.

Assign Profiles to Your Subscribers

The final step is to assign profiles to your subscribers using the Configuration Parameters screen. Click the Register Profiles button under the Registered Profiles list box and select the profiles used to format QDocs for this subscriber.

Fig. A.8
Register Profiles



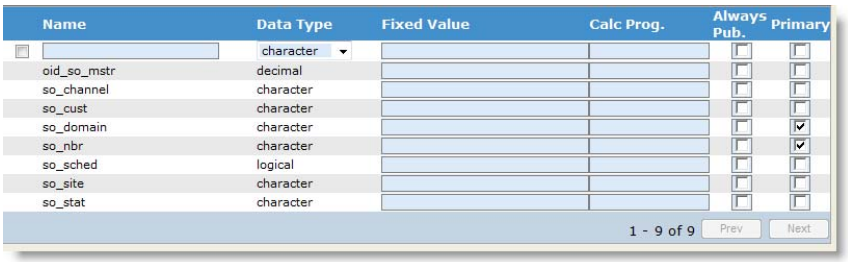
For details see “Modifying Profiles” on page 89.

Customizing Your Synchronization

If you want, once you have created or modified a business object, you can modify the related data objects to adapt them to your data synchronization requirements. For example, you can define custom fields to contain either a fixed value or a calculated value.

You use the Data Objects area of the Business Object screen to define a custom field as a fixed value or calculated field. You also can define this using the Profiles screen.

Fig. A.9
Fixed Value



Using Fixed Values

Defining a fixed value means that every time the business object is updated, that specified value is always used in that field.

When using fixed values, you can use subscriber tokens to handle the situation where each domain you are sending data to might need a specific value. For details, see “Configuring Subscriber Profiles” on page 45.

For details on using fixed values, see “QAD and Custom Business Objects and Profiles” on page 68 and “Modifying Business Object Data Objects” on page 80.

Using Calculated Fields

Calculated fields require a specific return value in a specific format, and correct connection parameters for the AppServer.

To define a custom field as a calculated field, in the Calculated Program field you enter the name of the Progress program (.p) to be run on the QAD QXtend AppServer for the associated source application (use the AppServer parameter settings you specified on page 25). The program determines the field to be used in the business object based on the destination domain.

For details on using calculated fields, see “Implementing Calculated Fields” on page 63.

Appendix B

Parameter Data

This section contains parameter data and other information about operation programs, calculated fields, the response parser, session context parameters, and inline triggers.

Operation Programs

QAD QXtend provides a set of standard operation elements to indicate the valid types of maintenance that can be requested for business objects. Allowed values are A (Add), M (Modify), R (Remove), and U (Unmodified). By using operation programs, you can customize these operation elements as required. For example, instead of using “A” as the operation element, you could use “Add.”

To create your operation programs, use the template program called `operationProgramTemplate.p` in the `<qxoserver>/src/samples` directory.

Operation programs require three procedures to be implemented:

- `getOperationTag` - Defines the name of the operation tag used for this profile. This procedure is required to support delta QDocs and to publish unchanged rows.
- `getOperationValue` - Defines the values of the operation tag used for this profile. This procedure is required to support delta Qdocs and to published unchanged rows.
- `updateOperationTags` - Compares two profile message datasets—the current message and the previous message—and updates the operation value of each record in the current message. The procedure returns an output parameter indicating whether or not there is a difference between the two messages. If there is no difference, the current message is discarded by the message publisher.

getOperationTag

```
procedure getOperationTag:
    define output parameter pcOperationTag as character.
    /*
    * Here the operation tag is called <action>
    */
    pcOperationTag = "action".
end procedure.
```

getOperationValue

```
procedure getOperationValue:
    define input parameter pcOperationValue as character.
    define output parameter pcCustomValue as character.

    /*
    * In this example, use New, Change, Delete and Unchanged as the values for the <action>
    tag
    */
    case pcOperationValue:
        when "A" then
            pcCustomValue = "New".
        when "M" then
            pcCustomValue = "Change".
        when "R" then
            pcCustomValue = "Delete".
        otherwise
            pcCustomValue = "Unchanged".
    end case.
end procedure.
```

getOperationTags

```
procedure updateOperationTags:
   define input parameter dataset-handle phCurrentMessage.
```

```
define input parameter dataset-handle phPreviousMessage.
define output parameter plModified as logical initial false.
```

Session Context Parameters

The following tables describe session context parameters required by SOAP messages for QXI and QXO. For details about session context information in QDocs, see “Session Context” on page 268.

QXtend Inbound

Table B.1
QXI Session Context Parameters

Qualifier	Name	Notes
QAD	domain	Domain that the message is for.
QAD	scopeTransaction	Scope the message to an entire transaction, back it out if there is an error (true or false).
QAD	version	Version of the QDoc.
QAD	mnemonicsRaw	Use the language mnemonic rather than the translated version (true or false).
QAD	username	User name to log in to the receiver application.
QAD	password	Password to log in to the receiver application.
QAD	action	Action code for Financials Objects.
QAD	entity	Entity code for Financials Objects.
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).
QAD	sessionID	QAD Enterprise Application session ID for authentication.

QXtend Outbound

BusinessObjectManager.p

Table B.2 shows session context parameters for `com/qad/qxtend/si/BusinessObjectManager.p`.

Table B.2
QXO Session Context Parameters

API	Qualifier	Name	Notes
createBusinessObjectMessage (DDP API)	QAD	domain	Domain the message is from.
	QAD	entity	Entity the message is from.
	QAD	businessEvent	The event that triggered the message.

API	Qualifier	Name	Notes
	QAD	email	Email address(es) of person requiring notification of status of message after Subscriber receives 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).
	QAD	sender	Sender ID of originating application.
	QAD	sourceApplication	Source Application code of originating application.
	QAD	delete	Message is for a delete operation (true or false).

Inline Triggers

QXtend provides in-line triggers for some business objects. You can define your own inline triggers for your application as required.

To create an inline trigger:

1. At the top of the program, add the `qxodef.i` include file to the variable definitions section at the top of the program.
2. Determine all the places in the code that require an event to be raised. Add a call to the `fireOutboundEvent` procedure. The parameters are different depending on whether you are using QAD SE or above, or MFG/PRO eB2 or below:
   - QAD SE and above

     ```
     run fireOutboundEvent (input 'SupplierMaintenance',
         input 'vd_mstr',
         input rowid(vd_mstr),
         input oid_vd_mstr,
         input 'DELETE').
     ```

     The parameters are:
     - Event name (character)
     - Table name (character)
     - Rowid of the record (rowid)
     - OID of the record (decimal)
     - Event type, either WRITE or DELETE (character)
   - MFG/PRO eB2 and below

     ```
     run fireOutboundEvent (input 'SupplierMaintenance',
         input 'vd_mstr',
         input rowid(vd_mstr),
         input 'DELETE').
     ```

     The parameters are:

- Event name (character)
- Table name (character)
- Rowid of the record (rowid)
- Event type, either WRITE or DELETE (character)

Appendix C

Response Parser

This section contains information on response parser.

Overview

You can create a customized response parser for a Web Service subscriber. The response parser must be implemented as an OpenEdge Object Oriented class that conforms to a standard interface that QXtend understands. The job of the response parser is to determine the status of the response in terms of success or failure, and in the case of failure, to extract the descriptions of the error from the response to make them available in the QXtend Message Monitor.

Requirements

The only requirement of a response parser is to implement an interface defined by QXtend Outbound. The class containing the interface is com.qad.qxtend.qxo.ISubscriberResponseParser. The definition for this interface is:

```
method public void parseResponse (input pcResponse as longchar,
                                  output dataset for dsQdocResponseDetails).
```

The definition of the dsQdocResponseDetails dataset is defined in the file `com/qad/qxtend/qxo/dsResponseDetails.i`.

```
define temp-table ttProcessingResult no-undo
   field resultStatus   as character
   field resultSequence as integer
   index resultSequence is primary unique resultSequence.
define temp-table ttProcessingException no-undo
   field exceptionNumber      as character
   field exceptionDescription as character
   field exceptionSeverity    as character
   field exceptionContext     as character
   field resultSequence       as integer
   field exceptionSequence    as integer
   index exceptionSequence is primary unique resultSequence exceptionSequence.
define dataset dsQdocResponseDetails
   for
      ttProcessingResult, ttProcessingException
      data-relation for ttProcessingResult, ttProcessingException
         relation-fields (resultSequence, resultSequence).
```

The response parser class receives the response as a longchar (pcResponse). This longchar variable contains the entire response, and it is up to the parser to determine from this variable whether the request was a success or failure, if it was a failure what type of failure, and any other information that can be gleaned.

Therefore, a template for creating a response parser would be:

```
using com.qad.qxtend.qxo.QdocResponseHandler.
class <classname> implements com.qad.qxtend.qxo.ISubscriberResponseParser:
   {com/qad/qxtend/qxo/dsResponseDetails.i}
   method public void parseResponse (input pcResponse as longchar,
                                     output dataset for dsQdocResponseDetails):
      < implement parsing code here >
   end method.
end class.
```

Populating the Response Dataset

The result of the parser must go in the dsQdocResponseDetails dataset. This dataset contains two temp-tables, ttProcessingResult and ttProcessingException. The ultimate result of the request must go in the ttProcessingResult temp-table in the field resultStatus. The values that this field can take are defined as static properties in the com.qad.qxtend.qxo.QdocResponseHandler class.

Property	Value	As Code
PendingStatusCode	PEND	QdocResponseHandler:PendingStatusCode
WarningStatusCode	WRN	QdocResponseHandler:WarningStatusCode
SuccessStatusCode	DLV	QdocResponseHandler:SuccessStatusCode
SendingErrorCode	SENDERR	QdocResponseHandler:SendingErrorCode
SoapFaultErrorCode	SOAPERR	QdocResponseHandler:SoapFaultErrorCode
ApplicationErrorCode	APPERR	QdocResponseHandler:ApplicationErrorCode

For example, if the parser determines that an application error has occurred, the code required would be:

```
create ttProcessingResult.
assign
   ttProcessingResult.resultStatus = QdocResponseHandler:ApplicationErrorCode
   ttProcessingResult.resultSequence = 1.
```

If the parser found two errors with appropriate descriptions, the code to add these to the dataset would be:

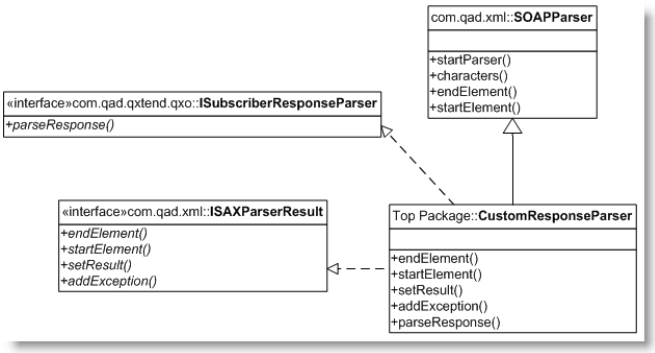
```
create ttProcessingException.
assign
   ttProcessingException.resultSequence = 1
   ttProcessingException.exceptionSequence = 1
   ttProcessingException.exceptionNumber = "999"
   ttProcessingException.exceptionDescription = "Part does not exist"
   ttProcessingException.exceptionSeverity = "ERROR"
   ttProcessingException.exceptionContext = "part = 'ABC123'".
create ttProcessingException.
assign
   ttProcessingException.resultSequence = 1
   ttProcessingException.exceptionSequence = 2
   ttProcessingException.exceptionNumber = "888"
   ttProcessingException.exceptionDescription = "User not authorized"
   ttProcessingException.exceptionSeverity = "ERROR"
   ttProcessingException.exceptionContext = "user = 'lchild'".
```

The resultSequence value must be the same as that in ttProcessingResult, and each ttProcessingException record must have a unique exceptionSequence number. The remaining fields can be filled out with information gleaned from the response, or left blank.

Parsing SOAP Responses

QXtend Outbound already parses SOAP responses from QXtend Inbound. As such, there is a class available to check for SOAP errors and this can be extended to parse the remainder of a SOAP response.

The structure that a class would take to parse a SOAP response would be as follows:



In other words, the CustomResponseParser will inherit the SOAPParser class, and implement the ISubscriberResponseParser and ISAXParserResult interfaces. This results in five methods that must be implemented in the parser. The SOAP Parser uses SAX to parse the XML response.

The ISAXParserResult interface is this:

```
method public void startElement ().
method public void endElement ().
method public void setResult (input pcResultCode as character).
method public void addException (input pcNumber      as character,
                                 input pcDescription as character,
                                 input pcSeverity    as character,
                                 input pcContext     as character).
```

The startElement method is called when a start node is encountered in the XML.

The endElement method is called when an end node is encountered in the XML.

The setResult method is called when the parser has determined the overall state of the response, where it will create a ttProcessingResult record.

The addException method is called when the parser has found a detailed exception message, where it will create a ttProcessingException record.

In addition to this, the SOAPParser class has a number of properties that can be used to assist in parsing the response.

Property	Type	Description
CurrentElementName	character	The current node in the XML
CurrentElementData	character	The current data between the nodes
CurrentNamespace	character	The namespace of the current node
CurrentAttributes	handle	Handle to the attributes of the current node
StopParser	logical	Flag to stop parsing at any time
ParserResultHandler	ISAXParserResult	Object reference to the result handler

Example A successful SOAP response from an external web service looks like this:

```
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd=
"http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <soapenv:Body>
    <Response xmlns="urn:custom-service:custom">
      <resultCode>
        <codeNumber>200</codeNumber>
        <userMessage>OK</userMessage>
```

```
        <debugInfo />
      </resultCode>
    </Response>
  </soapenv:Body>
</soapenv:Envelope>
```

And an unsuccessful response looks like this:

```
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd=
"http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <soapenv:Body>
    <Response xmlns="urn:custom-service:custom">
      <resultCode>
        <codeNumber>411</codeNumber>
        <userMessage>Login Failed</userMessage>
        <debugInfo>Unauthorized access</debugInfo>
      </resultCode>
    </Response>
  </soapenv:Body>
</soapenv:Envelope>
```

From this we can determine that a codeNumber of 200 means the request was successful, but any other code means the request returned an error. The response is also a SOAP response, hence there may be SOAP errors such as invalid address information, bad envelopes or other errors. We can extend the SOAPParser to create a response parser for this type of response. The response parser we will create will examine the codeNumber value, the userMessage value, and stop parsing once it has reached the end of the resultCode block.

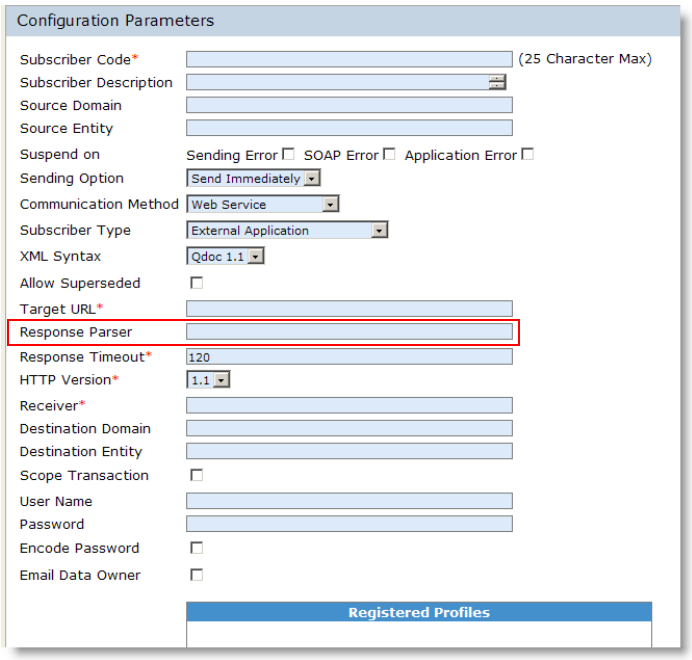
```
using com.qad.xml.SOAPParser.
using com.qad.xml.ISAXParserResult.
using com.qad.qxtend.qxo.ISubscriberResponseParser.
using com.qad.qxtend.qxo.QdocResponseHandler.

class CustomResponseParser inherits SOAPParser
                           implements ISAXParserResult,ISubscriberResponseParser:

   {com/qad/qxtend/qxo/dsResponseDetails.i}

   define variable gcReturnCode        as character no-undo.
   define variable gcExceptionDesc     as character no-undo.
   define variable giExceptionSequence as integer   no-undo.

   constructor CustomResponseParser ():

      /*
       * The SOAPParser class makes calls to a class that implements the
       * ISAXParserResult interface whenever it starts and finishes
       * processing a node. Make this object that class
       */
      assign
         ParserResultHandler = this-object.

   end constructor.

   method public void parseResponse (input pcResponse as longchar,
                                     output dataset for dsQdocResponseDetails):
   /*------------------------------------------------------------------------
      Purpose      : Starts the parsing of the Response XML

      Parameters   : [input]
                        pcResponse - The full XML response
                     [output]
                        dsQdocResponseDetails - Result and Exceptions dataset

      Notes        : Implementation of the ISubscriberResponseParser interface
    ------------------------------------------------------------------------*/
```

```
   dataset dsQdocResponseDetails:empty-dataset().

   /*
    * Call the method on the SOAPParser class to start parsing
    */
   startParser(input pcResponse).

end.

method public void startElement ():
/*------------------------------------------------------------------------
   Purpose      : Executes when a start node is encountered

   Parameters   : [none]

   Notes        : Implementation of the ISAXParserResult interface. There is
                  nothing we need to do when the parser encounters the
                  start of an element.
 ------------------------------------------------------------------------*/

end method.

method public void endElement ():
/*------------------------------------------------------------------------
   Purpose      : Executes when an end node is encountered

   Parameters   : [none]

   Notes        : Implementation of the ISAXParserResult interface.
 ------------------------------------------------------------------------*/

   if CurrentNamespace = "urn:custom-service:custom" then do:

      case CurrentElementName:
         when "codeNumber" then
            gcReturnCode = CurrentElementData.
         when "userMessage" then do:
            gcExceptionDesc = CurrentElementData.
            /*
             * The '200' response means there were no exceptions. Anything
             * else indicates there was a problem and we must log it. Set
             * the result to be an "APPERR"
             */
            if gcReturnCode <> "200" then do:
               setResult (input QdocResponseHandler:ApplicationErrorCode).
               addException (input gcReturnCode,
                             input gcExceptionDesc,
                             input "",
                             input "").
            end.
         end.
         when "resultCode" then do:
            if gcReturnCode = "200" then do:
               /*
                * Set the result to be a success, or "DLV"
                */
               setResult (input QdocResponseHandler:SuccessStatusCode).
               /*
                * Stop parsing at this point, since we have reached the
                * end node containing all the information we need
                */
               StopParser = true.
            end.
         end.
      end case.

   end.

end method.

method public void setResult (input pcResultCode as character):
/*------------------------------------------------------------------------
```

```
      Purpose      : Creates a processing result record

      Parameters   : [input]
                     pcResultCode - A string representing the result of the
                                    response

      Notes        : Implementation of the ISAXParserResult interface. This
                     method will allow the SOAPParser class to create
                     SOAPERR results
    ------------------------------------------------------------------------*/

      find first ttProcessingResult no-error.
      if not available ttProcessingResult then
         create ttProcessingResult.

      assign
         ttProcessingResult.resultStatus = pcResultCode
         ttProcessingResult.resultSequence = 1.

   end method.

   method public void addException (input pcNumber      as character,
                                    input pcDescription as character,
                                    input pcSeverity    as character,
                                    input pcContext     as character):
   /*------------------------------------------------------------------------
      Purpose      : Creates an exception record

      Parameters   : [input]
                     pcNumber      - Exception number
                     pcDescription - Exception description
                     pcSeverity    - Exception severity
                     pcContext     - Exception context

      Notes        : Implementation of the ISAXParserResult interface. This
                     will allow the SOAPParser class to log detailed
                     error messages when it gets a SOAP Error
    ------------------------------------------------------------------------*/

      create ttProcessingException.
      assign
         ttProcessingException.resultSequence       = 1
         ttProcessingException.exceptionSequence    = giExceptionSequence
         ttProcessingException.exceptionNumber      = pcNumber
         ttProcessingException.exceptionDescription = pcDescription
         ttProcessingException.exceptionContext     = pcContext
         ttProcessingException.exceptionSeverity    = pcSeverity.

      giExceptionSequence = giExceptionSequence + 1.
   end method.
end class.
```

To enable a subscriber in QXtend Outbound to use this response parser, just enter the full name of the parser class. In this case the parser class is just CustomResponseParser, however you can also put it in a package form; for example, com.xyz.parser.CustomerResponseParser.

Fig. C.1
Subscriber Configuration Parameters



The class must be compiled into the regular location for the QXtend Outbound services; that is, a directory in the PROPATH defined in the `start-sess.sh` script.

Glossary

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

Application Program Interface (API). A set of routines, protocols, and tools that connect applications, usually for the purpose of sharing data. The QAD QXtend interoperability framework removes the need for specially coded APIs.

B2B. Business-to-business. The exchange of products, services, or information between businesses, rather than between businesses and consumers.

Business Object. A set of tables and fields defined within QXO used to extract data from a source application.

CRUD. Create-read-update-delete. Used to describe a software application that can perform those actions on database records.

Direct Data Publish (DDP). A mechanism whereby raw business object event data is passed to QXO and directly published by the message publisher. This is usually a “push” mechanism in which the source application pushes the data for publishing, rather than the traditional “pull” mechanism, in which QAD QXtend controls the retrieval of information from the source application.

ebXML. See *Electronic Business XML (ebXML)*.

Electronic Business XML (ebXML). A joint project by the United Nations body for Trade Facilitation and Electronic Business Information Standards (UN/CEFACT) and the Organization for the Advancement of Structured Information Standards (OASIS) to use XML to standardize the secure exchange of business data.

Emulation. See *Terminal Emulation.*

Encryption. Conversion of data into a form that cannot be easily intercepted by unauthorized people.

Event. A change to data in a targeted source application that is of interest to one or more subscribers to QXO. A notification of each relevant change is written to the `qxevents` database where QXO polls for it.

Event Service. A process that runs on the QXO server that polls the `qxevents` database for change notifications. When a change is encountered, the event service queries the source application database for the changed data and writes the data as a raw message to the `qxodb` database.

Events File. A file generated during QDoc creation by QGen that contains field navigation information. Controls the processing of a QDoc request through the QAD Enterprise Applications user interface by indicating iteration levels and non-standard navigation. Each supported calling procedure has an associated events file.

Extensible Markup Language (XML). A specification designed especially for Web documents that allows the definition, transmission, validation, and interpretation of data between applications and organizations.

Extensible Style Language (XSL). A language for formatting an XML document; for example, showing how the data described in the XML document should be presented in a Web page.

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

4GL. An abbreviation for fourth-generation programming language, such as the Progress language. Direct APIs to the Progress code are more efficient than calls to the user interface

HTTP (Hypertext Transfer Protocol). The set of rules for exchanging text, graphic images, sound, video, and other multimedia files on the World Wide Web.

Iteration. A repeatable data entry cycle in QAD Enterprise Applications, such as sales order lines, or the entire sales order. Defined in QGen as the cycle from the first field back to the first field.

Java. An object-oriented programming language created by Sun Microsystems. Java is a device-independent language. Programs compiled in Java can be run on any computer. Java programs can be run as free-standing applications or as applets placed on a Web page.

Java 2 Platform, Enterprise Edition (J2EE). A recent release of Java designed to support the requirements of large-scale computing systems. Features include Java servlets and Java Server Pages (JSPs), which facilitate dynamic Web-enabled data access and manipulation.

Java Development Kit (JDK). A software development environment from Sun Microsystems for writing applets and applications in the Java programming language.

Java Message Service (JMS). An API from Sun Microsystems that supports formal communication—or messaging—between computers in a network.

Java Plug-in. Software provided by Sun Microsystems that replaces the default virtual machine associated with a Web browser. Using the Java plug-in allows developers to deploy Java applets that depend on the latest features of the Java platform and be assured that their applets will run reliably and consistently in both Microsoft Internet Explorer and Netscape Navigator.

Java Runtime Environment (JRE). A subset of the Java Development Kit for end users and developers who want to redistribute the Java runtime environment. The Java runtime environment consists of the Java virtual machine (JVM), the Java core classes, and supporting files.

Java Server Page (JSP). A technology for controlling the content or appearance of Web pages through the use of servlets, small programs that are specified in the Web page and run on the Web server to modify the page before it is sent to the user who requested it.

Java Virtual Machine (JVM). The part of the Java runtime environment responsible for interpreting bytecode.

JDK. See *Java Development Kit (JDK)*.

JRE. See *Java 2 Platform, Enterprise Edition (J2EE)*.

JSP. See *Java Server Page (JSP)*.

JVM. See *Java Virtual Machine (JVM)*.

Message Publisher. A process that runs on the QXO server that polls the `qxodb` database for raw messages. When one is encountered, the message publisher republishes the message in QDoc format. It uses the profiles to create the QDocs required by each interested subscriber. The QDocs are written back to the `qxodb` database. If no profile messages are created from a raw message, Message Publisher deletes the raw message from the `qxodb` database.

Message Sender. A process that runs on the QXO server that polls the `qxodb` database for QDocs. When one is encountered, the message sender sends it to each interested subscriber.

Namespace. In XML, a unique identifier for a collection of element type and attribute names. In an XML document, any element type or attribute name can have a two-part name consisting of the name of its namespace and then its local—or functional—name.

Profile. A subscriber-specific definition of a subset of a business object. Default profiles from QAD are loaded into the system and can be copied and modified to customize them.

PROPATH. An environment variable containing the list of directories Progress searches when looking for a program to execute.

QDoc. An inbound (to QAD Enterprise Applications) data document in XML format that conforms to generated schemas and events files from the QGen utility.

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

QGen. A tool that captures field and navigation information for an QAD Enterprise Applications or other Progress data entry program, and generates QDoc schemas and events from the program data.

Queue Manager. An optional interface to manage QDoc requests and responses through a specified directory structure. The Queue Manager can also add a SOAP envelope to a QDoc to support QAD QXtend requirements.

qxevents Database. A database added to the QAD Enterprise Applications source application where change notifications are written after a data update in the source application.

qxodb Database. The QXO database, installed on the QXO server. This database contains raw messages, QDocs, and log records.

QAD QXtend Inbound (QXI). The companion to QXO for QAD applications, this product accepts QDocs from external sources and enters them into an QAD Enterprise Applications—or other QAD product—database.

QXtend Message Monitor. A .NET UI-based plug-in for QXO that tracks the outbound profile messages that are sent to external subscribers, as well as subscriber responses. Using the QAD QXtend Message Monitor allows the lifecycle of messages to be traced through QAD QXtend.

Receiver. A receiver identifies QAD Enterprise Applications instances that process inbound QDoc requests.

Remote Procedure Call (RPC). A protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details.

Script. A program or sequence of instructions that is interpreted or carried out by another program.

Secure Sockets Layer (SSL). A program layer for managing the security of message transmissions in a network. The program layer exists between an application (such as a Web browser or HTTP) and the Internet's TCP/IP layers. Sockets refers to the sockets method of passing data back and forth between a client and a server program in a network or between program layers in the same computer.

Service Interface Layer. An API that allows data originating from a source application type that uses DDP to be input into a target application. The service interface adapter API sends QDocs to a Progress program connected to the service interface layer. The service interface layer is the QAD standard for Progress-application-to-Progress-application communication without requiring QAD QXtend or any third-party message handler.

Servlet. Programs—similar to Java applets—that run on the server (rather than the client) and are used to run interactive Web applications.

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

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

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

Source Application. Any Progress-based production system set up with triggers and a `qxevents` database to notify QXO of a relevant change.

SSL. See *Secure Sockets Layer (SSL)*.

Subscriber. A destination for QDocs defined in QXO. The destination can be either a Web service URL or a directory location.

TCP/IP. See *Transmission Control Protocol/Internet Protocol (TCP/IP)*.

Telnet. A user command and underlying TCP/IP protocol that lets you access applications and data on remote—or *host*—computers.

Terminal Emulation. Use of a personal computer to interact with a computer that uses a different operating system. The terminal emulation program runs as a separate task with its own window. The application interface presented in this window is character-based or text-only.

Tomcat. The servlet container used in the official reference implementation for the Java Servlet and Java Server Pages (JSP) technologies. Tomcat is developed in an open and participatory environment and released under the Apache Software Foundation license.

Transmission Control Protocol/Internet Protocol (TCP/IP). The basic communication language or protocol of the Internet. It can also be used as a communications protocol for intranets and extranets.

Triggers. A program defined in the database of a source application that automatically records a change to specific tables whenever a change occurs.

UI. See *User Interface (UI)*.

Uniform Resource Identifier (URI). A method of identifying or reserving a point of content on the internet, such as a page of text, a graphic image file, or a program. A URI typically describes the:

- Mechanism used to access the resource
- Specific computer that the resource is housed in
- Specific name of the resource (a file name) on the computer

The most common form of URI is a uniform resource locator (URL).

Uniform Resource Locator (URL). A text string that indicates the location of an intranet or Internet resource.

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

User Interface (UI). The portion of an application that is visible to the user, and the mechanism by which the end user interacts with the application, enters information into the application, and sees the results of the interaction.

UUID. See *Universal Unique Identifier (UUID)*.

WAR. See *Web Archive File (WAR)*.

Web Archive File (WAR). A compressed file containing a Web application and its related files. Assists in easily deploying an entire application.

Web Services. Vendor-neutral, XML-based, remote procedure call (RPC) protocol that allows any system to run programs in other, dissimilar systems.

Web Services Description Language (WSDL). An XML-based language used to describe the services that a business offers and to provide a way for individuals and other businesses to access those services electronically.

World Wide Web Consortium (W3C). An international industry consortium that seeks to promote standards for the evolution of the Web and interoperability among Internet products by producing specifications and reference software.

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

XML Schema Definition (XSD). An abstract representation of the elements in an XML document that can be used to verify that each item of content adheres to the associated element's description. The XSD standard follows the W3C recommendation.

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

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

Index

R