Chapter 18

QGen

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

Introduction ***228***
Discusses program structure and terminology, QGen files, and generating native APIs.

Starting QGen ***232***
Explains how to start QGen with a startup script.

Mapping a Program for Regular UIAPI Interface Programs ***232***
Explains how to map a program, map first entry events, map iterations, and map comments.

Map Comments ***238***
Explains how to save, load, change mode to Update, change mode to Run Through, New Run Through, and generate docs.

Troubleshooting QGen ***245***
Explains how to troubleshoot QGen.

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 241 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 230.

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. During installation, only standard QGen master data (`.dat`) files that support the current QAD Enterprise Applications release are deployed. If a `.dat` file does not match the screen (usually because of screen customization), you can use QGen to remap the screen and save the customized `.dat` file without overwriting the standard one.

When you install QXtend, the installer now copies the QGen `.dat` files for the current release of QAD Enterprise Applications to this location:

`<QXtend adapter>/tools/datFiles/standard/`

When you modify the `.dat` files and do not specify a directory when saving them, customized `.dat` files are always saved to the `<QXtend adapter>/tools/datFiles/` directory, leaving the standard `.dat` files intact.

When you load `.dat` files without specifying a directory, the system first looks for the files under `<QXtend adapter>/tools/datFiles/`. If the files cannot be found under this directory, the system then loads the standard `.dat` files under `<QXtend adapter>/tools/datFiles/standard/`.

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, see *QAD .NET UI Administration Guide*.

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 254.

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 231.
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.

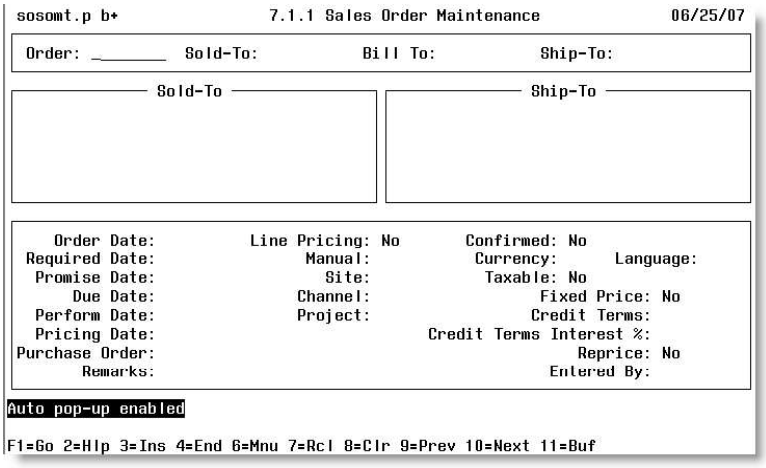
This starts an QAD Enterprise Applications session with QGen running in the background.

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

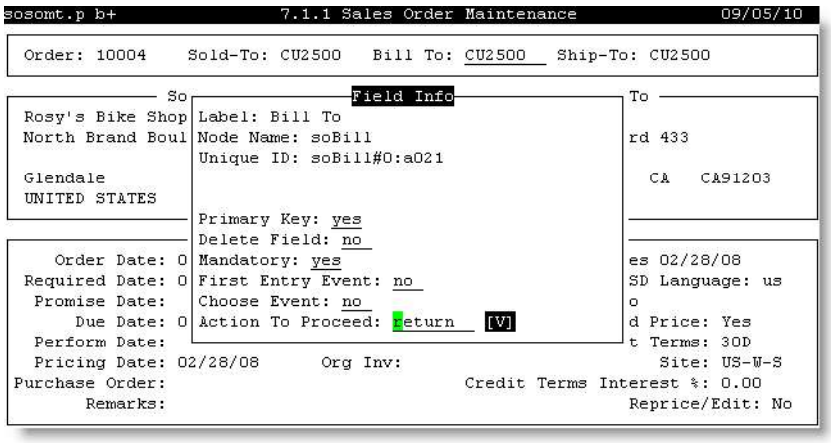


3 Enter data in the first field or select an existing record to edit and press Enter to move to the next field.

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 235.

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