



QAD Enterprise Applications
Standard Edition

Training Guide
Best Practices for Customizing
QAD Progress Applications

70-2946A
QAD 2008 Standard Edition
Database: Standard Edition - QMS
US
NOV 2008

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

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

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

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

Copyright ©2010 by QAD Inc.

BestPracticeCustomization_TG_v2008SE.pdf/QAD

QAD Inc.

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

Contents

About This Course	1
Course Description	2
QAD Web Resources	3
Chapter 1 Business Considerations	5
Business Considerations	7
Risks	8
Support Implications	9
Upgrade Implications	10
Review	11
Chapter 2 Architecture of QAD Enterprise Applications	13
Introduction	14
Components of QAD Enterprise Applications	15
QAD Enterprise Applications Versions	17
QAD Enterprise Applications Databases	18
QAD Enterprise Applications Directories	19
QAD Enterprise Applications eB Progress Licenses	20
External Interfaces	21
Advanced Planning and Optimization	22
Demand Planner	23
Sales and Use Tax	24
Warehousing Interface	25
Q/LinQ	26
Data Synchronization	28
Logistics	29
QAD Enterprise Applications Translation	30
Menu Structure	32
Menu System	33
Main Menu: Character	34
Main Menu: GUI	35
Main Menu: QAD Desktop	37
Menu Execution Options	38
Customizing Menus	39

Upgrading Menus	40
Security	41
Field Validations	42
Security Types	43
Overview of Security Types	44
Security Flow Diagram: eB and Below	45
Security Flow Diagram: eB2	46
Field Security	47
Custom Field Security	50
Password Validation	53
Site Security	54
Review	55
Chapter 3 Product Requirements	57
Introduction	58
Terminology	60
Development and Admin Products	63
QAD Enterprise Applications Product Requirements	64
QAD Enterprise Applications Source Code	65
QAD Enterprise Applications Available Source Code	66
Review	67
Chapter 4 PROGRESS Basics	69
Introduction	70
File Name Extensions	71
Comments	72
Syntax Elements	73
Data Types	74
Default Display Formats	75
Record Retrieval	76
Scoping	77
Record Scoping	78
Transactions	80
Record Locking	82
Frame Scoping	83
Named Frames	84
BREAK Group	85
Include Files	86
Internal Procedures	87
Passing Parameters	89
User-Defined Functions	90
TEMP-TABLES	91

Buffers	92
Preprocessor Directives	93
Review	94
Chapter 5 Functionality Reuse	95
Introduction	96
Terminology	97
Common Include Files	98
eB Common Include Files	100
mfdtitle.i	102
mfdeclre.i	107
Exercise 1	109
Global Variables	110
gprun.i	111
pxrun.i	112
mfnp.i	115
mfmsg.i	118
mfmsg03.i	121
pxmsg.i	122
gpselout.i	126
mfselprt.i	131
mfselbpr.i	135
mfrpchki.i	140
mfreset.i	141
mfquoter.i	142
mfphead.i and mfphead2.i	143
mfphead.i	145
mfrtrail.i	148
mfrtrail.i and mfrtl080.i	150
Exercise 2	151
Exercise 3	152
Field Help	153
Window Help	154
Record Sequencing	155
E-mail Definition Maintenance	157
User Maintenance	160
Event Notification	161
Review	162
Chapter 6 Database Schema	163
Introduction	164
Terminology	165

Standard Databases	166
QAD Enterprise Applications Data Definitions	168
QAD Enterprise Applications File Relationships	171
User-Definable Fields	173
User Workfile	174
Changing Database Schema	176
“Frozen” Databases	178
Adding New Database Schema	181
Schema Naming Conventions	183
File Naming Conventions	184
Index Naming Conventions	186
Adding Indexes	187
Adding Fields	192
Adding Files	197
Side Databases	198
Database Naming Conventions	201
Creating Databases	202
Adding Database Files	205
Database Definition Files	211
Creating Data Definition File	212
Dumping Data Definitions	213
Loading Database Definitions	216
Database Connection Options	219
CRC Issues	222
Field Validation	224
Exercise: Adding Schema:	227
Adding a Side Database	228
Summary	240
Chapter 7 Configuration Management	241
Introduction	242
Terminology	243
Vanilla QAD Enterprise Applications	244
Patched QAD Enterprise Applications Code	245
Development Environments	246
Data Management	247
Test Environments	248
Promotion of Code to Production	250
Verifying Issues	252
Managing Multiple Versions	253
Review	254

Chapter 8 Programming Standards 255

Introduction	256
Terminology	257
Naming Conventions	258
Naming Source Code	259
Naming Include Files	260
Naming Validation Procedures	261
Changing Standard Source Filenames	262
Patch Marking Conventions	265
Revision Control	266
Revision Level	270
Inquiry Program Template	271
Report Program Template	272
Maintenance Program Template	277
Net UI Infrastructure	280
Java UI	281
Programming Considerations	292
Database Tables	293
Database Fields	294
Database Indexes	297
GUI Converter	325
GUI Programming Considerations	340
Desktop 2 Development Standards	343
Exercise: 4 Desktop 2	372
Translations	374
Performance Considerations	375
Exercise: Custom Programming	403
Programming Lab	406
Programming Lab Solution	407
Summary	418

Chapter 9 Multi-Database Programming 419

Introduction	420
Alias Statement	421
Alias and Multiple Databases	423
Site Maintenance	426
gpalias.p	431
Summary	435

Chapter 10 Compiling Code.....437

Introduction	438
Logical Database Names	439

Compiling Using MFG/UTIL	442
Hands On Exercise: Compiling Lab	454
Review	473
Chapter 11 Triggers	475
Introduction	476
Terminology	477
Schema Triggers	478
CREATE, DELETE, FIND Headers	480
WRITE Trigger	484
Assign Trigger	487
Return Error	490
General Considerations	491
Review	493
Chapter 12 Browsers	495
Introduction	496
Look-up Browsers	497
Power Browse	498
Views	500
Browse Tables in GUI Database	501
Designing Custom Browsers	503
Naming Conventions	505
Generated Source Code	513
Linking Browsers	514
Designing Views	518
Filtering and Sorting Browsers	526
Print Browsers	531
Browse Graphing	533
Hands On Exercise: Browse Lab	536
Review	549
Chapter 13 Customizing On-Line Help	551
Introduction	552
Customizing QAD Enterprise Applications Help	553
Review	557
Chapter 14 CIM Interface	559
Introduction	560
CIM Mechanism	561
Rationale for Using CIM	562
CIM Data Formatting	563

CIM File Format	564
Building CIM Files	565
CIM Data File	568
CIM Testing	589
CIM Test Program	590
Loading CIM Data	592
Why Test CIM?	598
Exercise: CIM Lab	602
Review	606
Chapter 15 E-Mail	607
E-Mail Definition Maintenance	609
User Maintenance	611
Example	612
Internal Procedure	613
Event Notification	614
Other Possibilities	615
Example	616
Chapter 16 International Multi-Language Labels	619
General Translation Principles	621
Accessing Label Functions	624
Label Master Maintenance	626
Term Naming Conventions	628
Label Master Terms	629
Rules	631
Label Detail Maintenance	635
Hands On Exercise 5:	683
Appendix A Programming Standards: Sample Report Template ..	687
Appendix B Programming Standards: Sample Report Program ...	691
Appendix C How to Web-Enable a Report	697
Appendix D GUI Converter	711

About This Course

Course Description

This course describes the background knowledge, environment, methods, and limitations in customizing the QAD Enterprise Applications. This course covers:

- How the QAD Enterprise Application ERP system is built
- The basics of Progress
- Reuse of existing QAD Enterprise Applications programs
- Database schema
- Programming standards
- Multiple database programming
- Compilation considerations
- Schema triggers
- Browses
- CIM interface

Course Objectives

By the end of this class, students will learn:

- QAD Enterprise Applications architecture
- Basic product, business, and design requirements
- Programming standards and reusability of standard QAD Enterprise Applications code
- How to customize the schema, browses, help and other aspects of QAD Enterprise Applications
- How to use the CIM interface in customizations
- How to port your customizations to new versions of QAD Enterprise Applications

Audience

- Programmers
- System administrators
- Technical consultants
- Technical support specialists

Prerequisites

- Knowledge of Progress 4GL
- Basic knowledge of Progress Database Administration
- Relational Database Experience
- Application Programming Experience
- Basic understanding of QAD Enterprise Applications

Course Credit and Scheduling

This course is designed to be taught in 4 days

QAD Web Resources

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

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

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

Chapter 1

Business Considerations



Business Considerations

In this section, you learn how to:

- ▲ Processes and Procedures
- ▲ Reporting Requirements
- ▲ Customer Expectations
- ▲ Product Configuration

Business Considerations



Business Considerations

- ▲ Are business needs fully met
- ▲ Risks involved
- ▲ Support implications
- ▲ Upgrade implications

QAD Proprietary

BPC-01-BU-020

The most basic reason that people customize is that, regardless of how broad and deep the functionality of an application may be, including QAD Enterprise Applications, there are going to be some business needs that are not completely met by a standard, off-the-shelf application. Some examples of this include new reports that may be needed, additional functionality necessary to support unique business processes, and interfaces to other applications that are needed to support your business.

Business needs are not fully met by “standard” QAD Enterprise Applications:

- New reports are needed
- Additional functionality needed
- Interfaces to other applications

Risks


Our Passion. Your Advantage.

Risks Involved...

- ▲ QAD support implications
- ▲ Upgrade implications

QAD ProprietaryBPC-01-BU-040

There are, however, some risks you need to manage when undertaking any customization work. There are support and upgrade implications. We will talk about these risks and address some ways to mitigate them throughout this class.

Support Implications



Support Implications

- ▲ QAD supports standard QAD Enterprise Applications
- ▲ Code management
- ▲ Plain “vanilla” QAD Enterprise Applications test environment
- ▲ You must filter support calls
- ▲ End-user training
- ▲ Documentation
- ▲ On-line Help

QAD Proprietary

BPC-01-BU-050

These are some of the support risks that should be taken into consideration before customizing.

- QAD only supports standard product QAD Enterprise Applications. This will force some requirements on you to manage the code from the standard distribution of QAD Enterprise Applications and all customizations that you develop. To get help from QAD support, you will have to maintain the ability to reproduce any user-reported problems in a “standard” test environment. You will also have to make sure users understand what is standard and what is custom, or have all issues go through your IS group to filter what is your custom work.
- QAD spends a lot of time and money developing end-user training materials, documentation, and on-line help. By customizing, you risk obsoleting these resources for your end-users and you should consider how to mitigate these risks.

Upgrade Implications


Our Passion. Your Advantage.

Upgrade Implications

- ▲ Invasive customizations
- ▲ Version-specific customizations
- ▲ Why you pay maintenance

QAD Proprietary

BPC-01-BU-060

Another risk is getting version locked and not being able to upgrade to later versions of QAD Enterprise Applications when they become available. One of the benefits you get from purchasing maintenance is the availability of upgrades. Companies have invested a lot in purchasing, installing, and implementing QAD Enterprise Applications and will want to make sure that future upgrades are not more expensive in terms of migrating custom code than they need to be, or would be otherwise.

Review



Business Considerations

- ▲ The Key Business considerations before Customizing in QAD Enterprise Applications
 - Processes and Procedures
 - Reporting Requirements
 - Customer Expectations
 - Product Configuration

Chapter 2

Architecture of QAD Enterprise Applications

Introduction

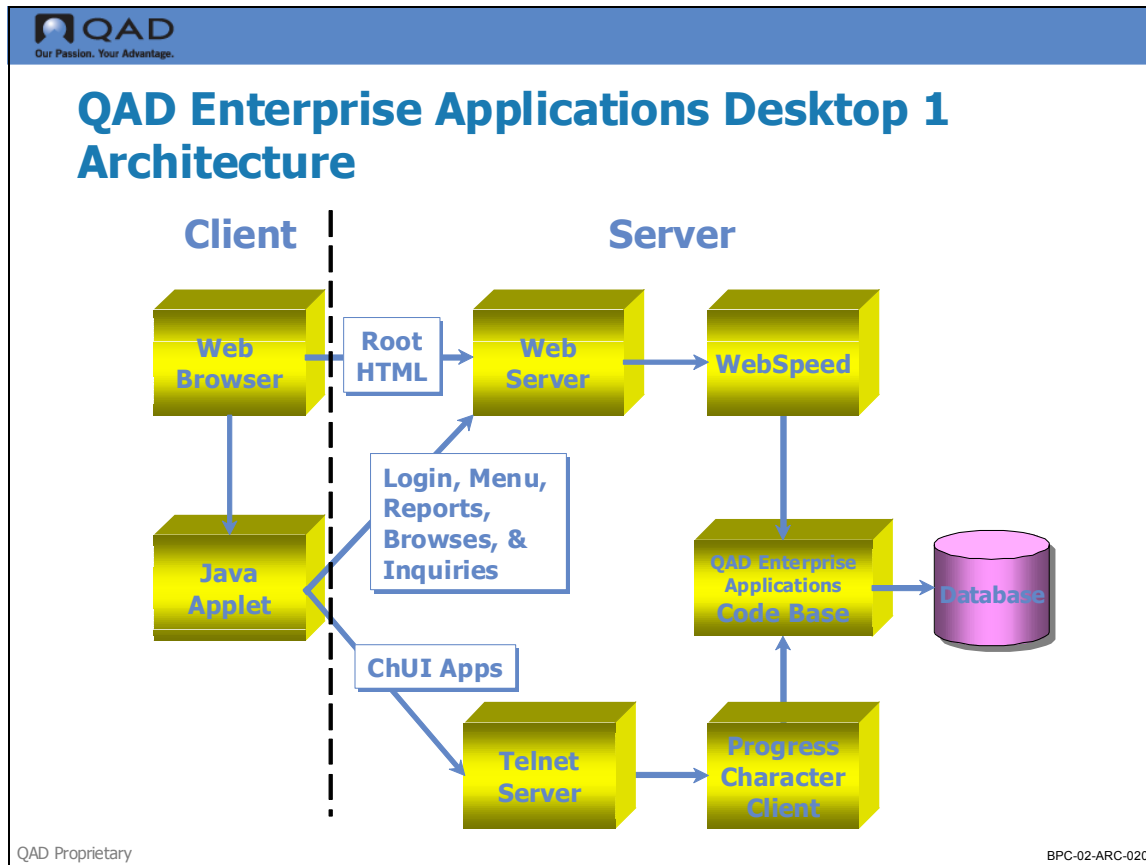


QAD Enterprise Applications Architecture

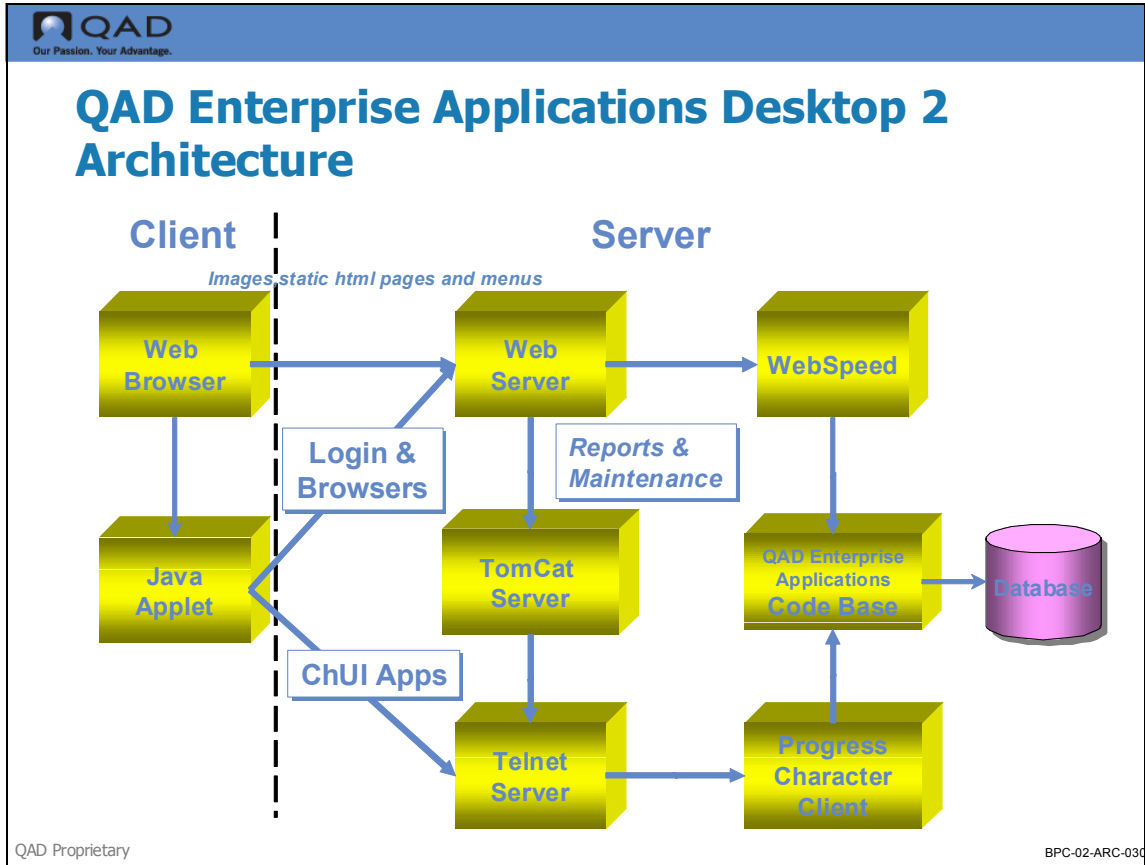
In this section, you learn how to:

- ▲ Identify the components of QAD Enterprise Applications
- ▲ Identify the menu structure of QAD Enterprise Applications
- ▲ Identify the security types in QAD Enterprise Applications

Components of QAD Enterprise Applications

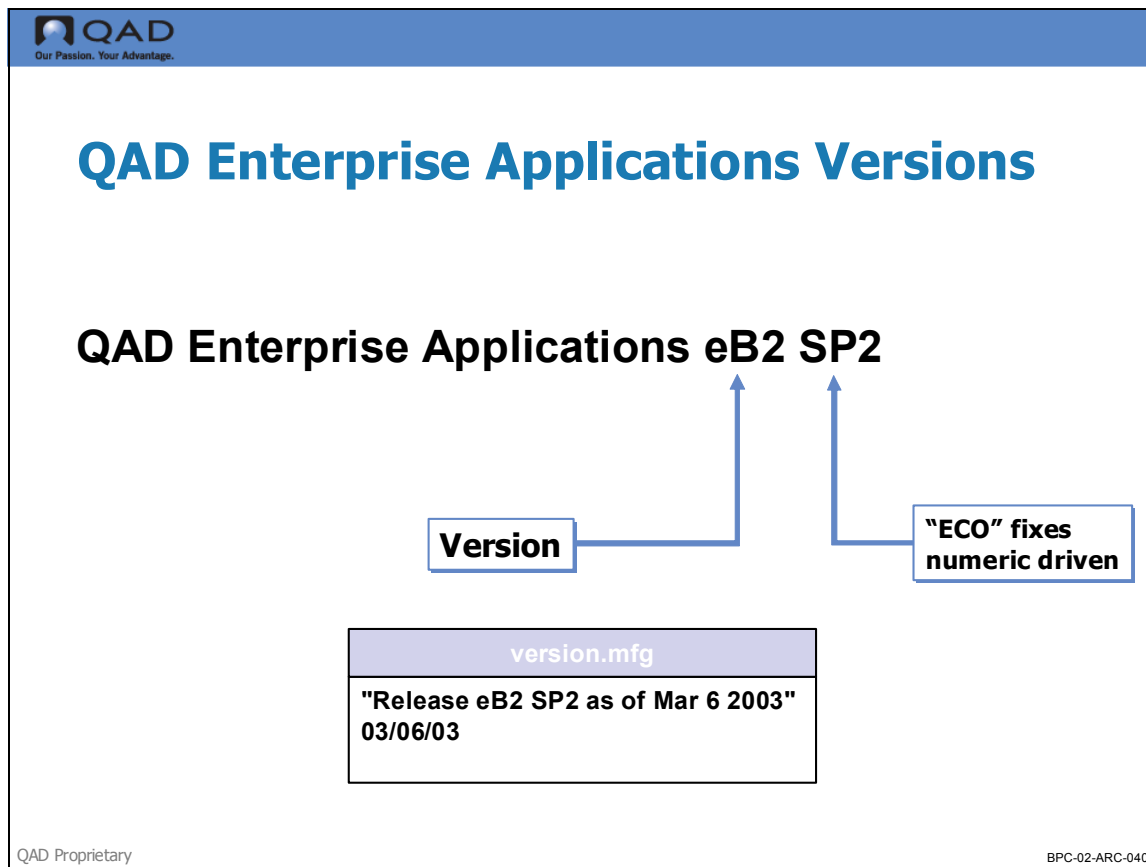


QAD Enterprise Applications Desktop 1 Architecture



QAD Enterprise Applications Desktop 2 Architecture

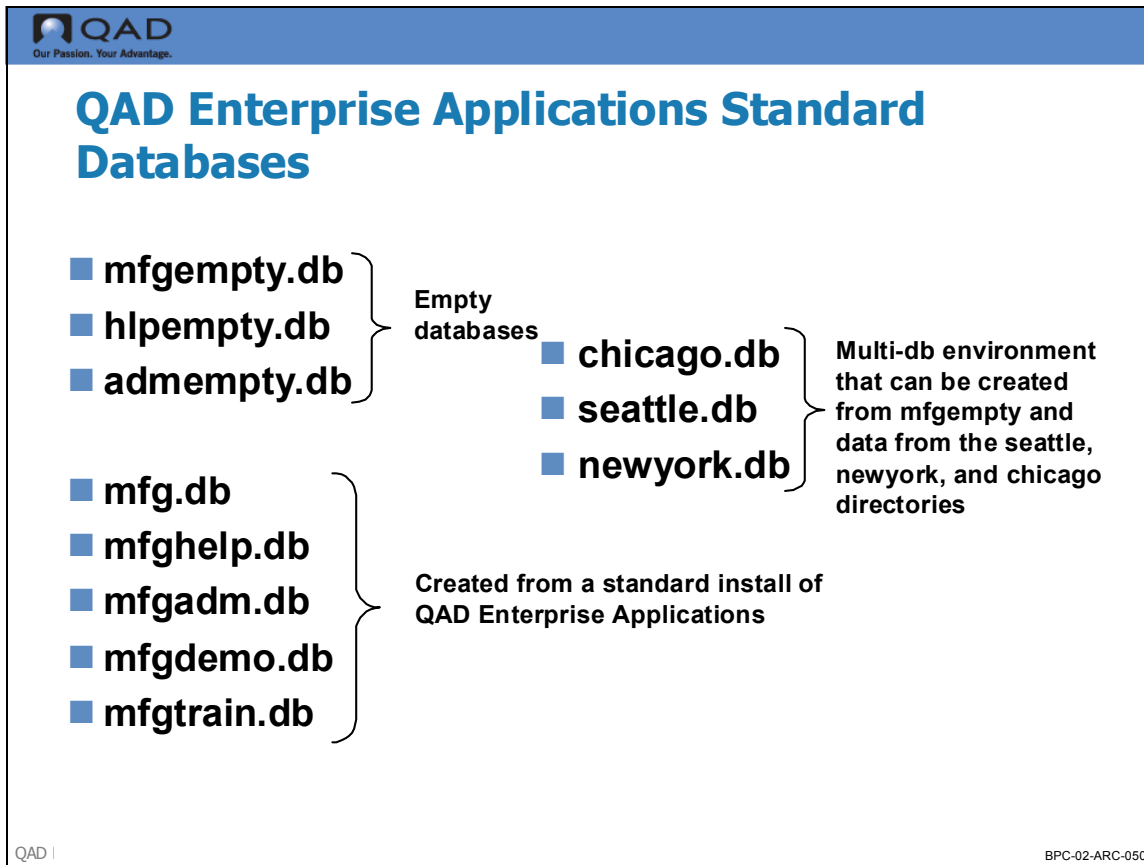
QAD Enterprise Applications Versions



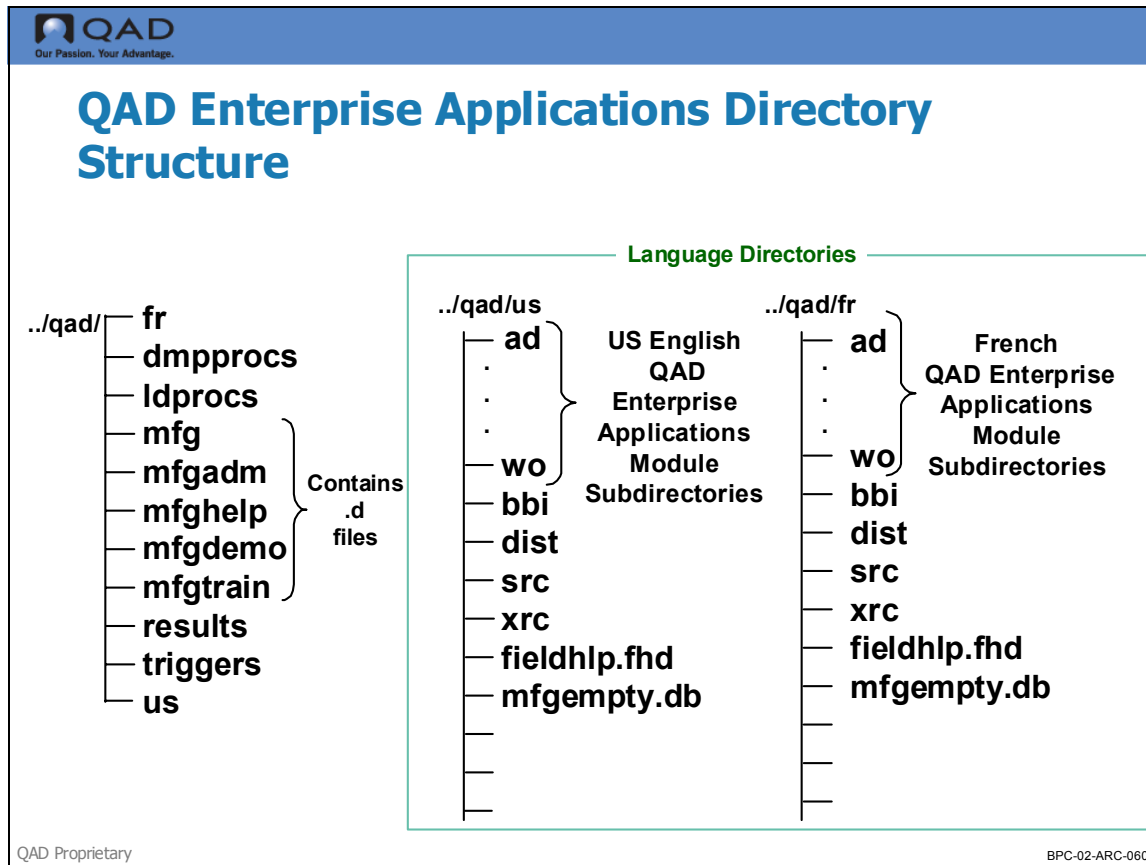
Recompile for new versions of Progress, generally there are not any functional changes.

Note With 8.6E, QAD started delivering enhancements within lettered releases. From eB (9.1), there are no more lettered releases. Service packs have taken on the role of delivering “bug” fixes.

QAD Enterprise Applications Databases



QAD Enterprise Applications Directories



QAD Enterprise Applications eB Progress Licenses


Our Passion. Your Advantage.

QAD Enterprise Applications Progress Licenses

- ▲ The Progress license file (progress.cfg) is located in the \$DLC and can be viewed by running \$DLC/bin/showcfg
- ▲ For QAD Enterprise Applications, the following licenses can be required
 - Workgroup/Enterprise DB server
 - Query/Results (for all Character and/or GUI users)
 - 4GL/Provision (only for the Administrator)
 - Client networking (only GUI)
 - WebSpeed (only NetUI/Desktop)
- ▲ Note for NT users, use %DLC% instead of \$DLC

QAD Proprietary
BPC-02-ARC-070

Example

Configuration File: /u1/progress/dlc91d01/progress.cfg

- Company Name: YourCompany
- Product Name: 4GL Development
- Installation Date: Mon Jul 1 12:05:04 2002
- User Limit: 64
- Expiration Date: None
- Serial Number: 009999999
- Control Numbers: YYYYYY - ZZZZZ - MMMMM
- Version Number: 9.1D
- Machine Class: KB
- Port Number: 11

External Interfaces



External Interfaces

- ▲ Advanced Planning and Optimization (36.5)
- ▲ Demand Planner
- ▲ Sales and Use Tax
- ▲ Warehousing
- ▲ Q/LinQ
- ▲ Data Synchronization
- ▲ Logistics

Advanced Planning and Optimization



Advanced Planning and Optimization

- ▲ The Advanced Planning and Optimization (APO) API integrates QAD Enterprise Applications with Factory Optimizer and Supply Chain Optimizer
 - This API exports Planning Data from QAD Enterprise Applications to Optimizer applications and imports Scheduling data back from Optimizer to QAD Enterprise Applications
- ▲ The APO available-to-promise (ATP) API determines real-time ATP information from Optimizer applications for a particular demand in QAD Enterprise Applications

Demand Planner



Demand Planner

- ▲ Demand Planner is an Oracle-based advanced demand management application developed by Demantra Ltd. which replaces the forecasting calculation functions of QAD Enterprise Applications
- ▲ The Demand Planner API provides a method of exchanging data between the two systems

Sales and Use Tax



Sales and Use Tax

- ▲ The Sales and Use Tax Interface (SUTI) is an application programming interface (API) designed for companies that use QAD Enterprise Applications and need advanced tax calculation solutions offered by companies like Vertex
- ▲ SUTI works in conjunction with Global Tax Management (GTM) and QUANTUM to retrieve tax jurisdiction information from the QUANTUM system and calculates US and Canadian regional taxes using QUANTUM calculation rules and tables

Warehousing Interface



Warehousing Interface

- ▲ Lets you transfer data between an externally controlled warehouse and QAD Enterprise Applications
- ▲ Creates import transactions that send inventory receipts, shipment/issues, adjustments, and dispositions processed in the warehouse to QAD Enterprise Applications
- ▲ Once they are mapped into the format required by the interface, you must load them into QAD Enterprise Applications using CIM interface
- ▲ The Warehousing interface is not an EDI solution
- ▲ Considered an extension of the QAD Enterprise Applications site. QAD Enterprise Applications manages its supply/demand orders, records its summary inventory balances, and sends it item, trading partner, and order information. In turn the warehouse system sends most of its inventory transactions back to QAD Enterprise Applications.

Q/LinQ



Q/LinQ (1 of 2)

- ▲ Is a complete tool set for building integrations with third-party or in-house applications for complex data exchange
- ▲ It is also the infrastructure for administering and managing the data exchange between QAD Enterprise Applications and the external applications such as logistics and distribution systems, factory automation systems, scheduling systems, and Sales order entry systems



Q/LinQ (2 of 2)

- ▲ It is a framework of interfaces for data coming into and going out of QAD Enterprise Applications. It receives data and passes it to its destination. Data can come from a file, an QAD Enterprise Applications process, or an application or a system external to QAD Enterprise Applications. The destination can be a file, another QAD Enterprise Applications process, or an application or system external to QAD Enterprise Applications.
- ▲ Q/LinQ can instruct the external application to process the data and then inform Q/LinQ if the processing succeeded, failed or generated errors

Data Synchronization



Data Synchronization

- ▲ DataSync synchronizes static data such as Item Master data among multiple, distributed QAD Enterprise Applications databases
- ▲ DataSync does not synchronize transaction data or entire databases
- ▲ The data is exported from or imported to each QAD Enterprise Applications database using Q/LinQ

Logistics



Logistics

- ▲ The Logistics application program interface supports data exchange between QAD Enterprise Applications and external logistics applications
- ▲ Logistics is built on the QAD Enterprise Applications Q/LinQ infrastructure
- ▲ Specifically designed to exchange data between logistics and supply-chain-management applications

QAD Enterprise Applications Translation



QAD Enterprise Applications Translation (1 of 2)

- ▲ Before QAD Enterprise Applications eB, for each language, a separate set of source was created / delivered
- ▲ As-of QAD Enterprise Applications eB (with SP1), only one “universal” (English) source is available and translations are done at run-time
 - Each translatable string in the source is called a term
 - For each term, 4 types of labels are available (Long, Medium, Short and Stacked)
 - For new (database) fields, “link” them to existing terms and avoid creation of new terms (and thus new translations)



QAD Enterprise Applications Translation (2 of 2)

- ▲ Database (admin)
 - lbl_mstr (term with translation)
 - lbl_det (link between fields & terms)
- ▲ Progress (software)
 - Translation done by glabel.p
- ▲ In QAD documentation, the Translation mechanism is also called Externalized Labels

Menu Structure



Our Passion. Your Advantage.

Menu Structure Overview

- ▲ Menu System
- ▲ Character Menu
- ▲ Graphical Menus
- ▲ Desktop Menu
- ▲ Menu Execution Options
- ▲ Customizing Menus
- ▲ Menu Upgrade Issues

QAD Proprietary

BPC-02-ARC-190

QAD Enterprise Applications features three user interfaces.

- QAD Enterprise Applications for Windows
- Character mode
- QAD Desktop

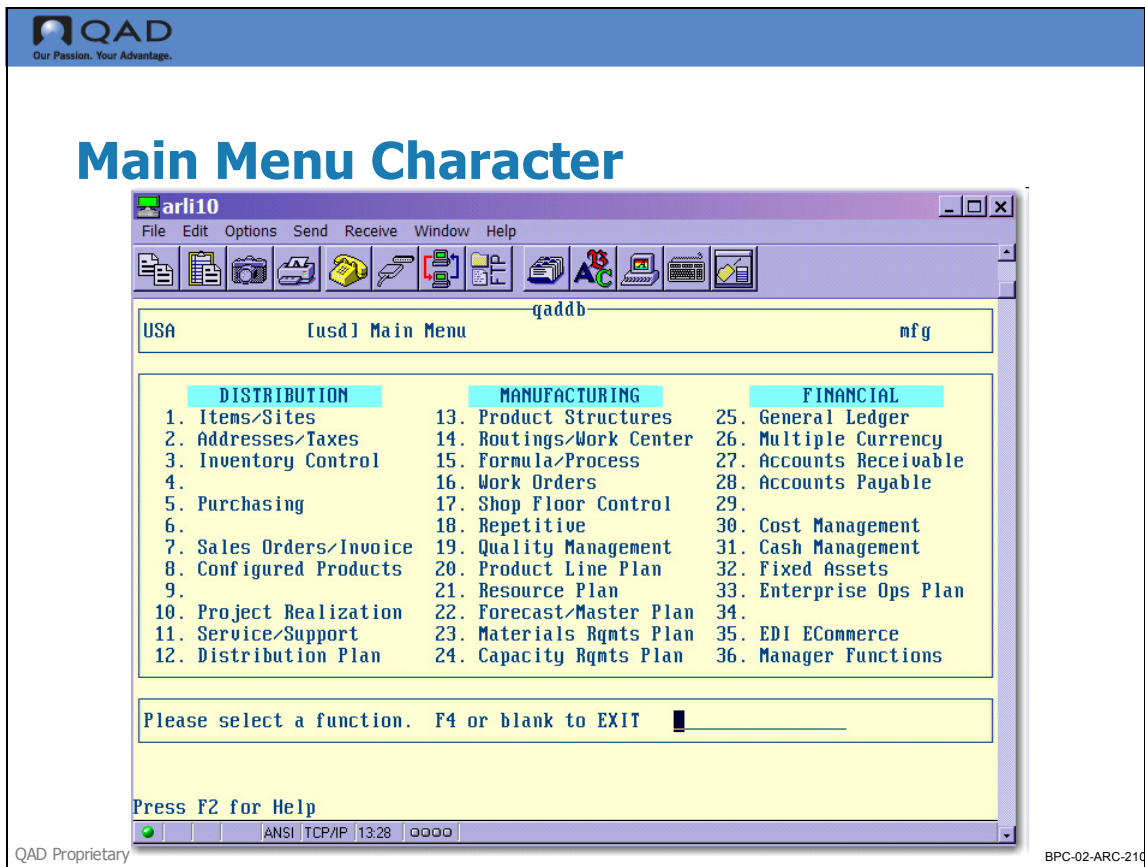
Menu System



Menu System

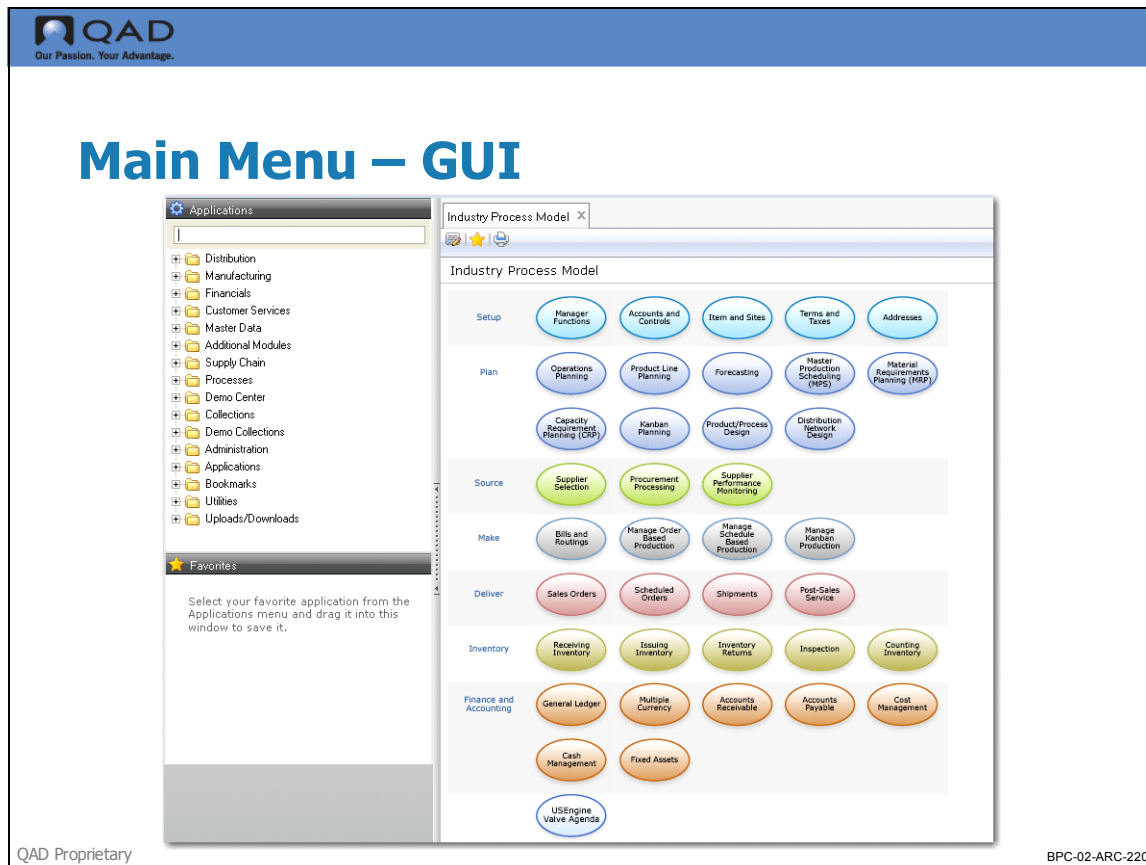
- ▲ Menu System controls the menu options displayed when the user logs in
- ▲ Use Menu System Maintenance to assign menu labels and execution files to menu numbers
- ▲ All the menu information is contained in the mnd_det table
- ▲ mnd_det is modified by Menu System Maintenance and User Function Maintenance
- ▲ Menu changes are visible immediately in the Character and Windows interfaces
- ▲ In Desktop, you must execute the Menu Build utility before changes can be viewed

Main Menu: Character



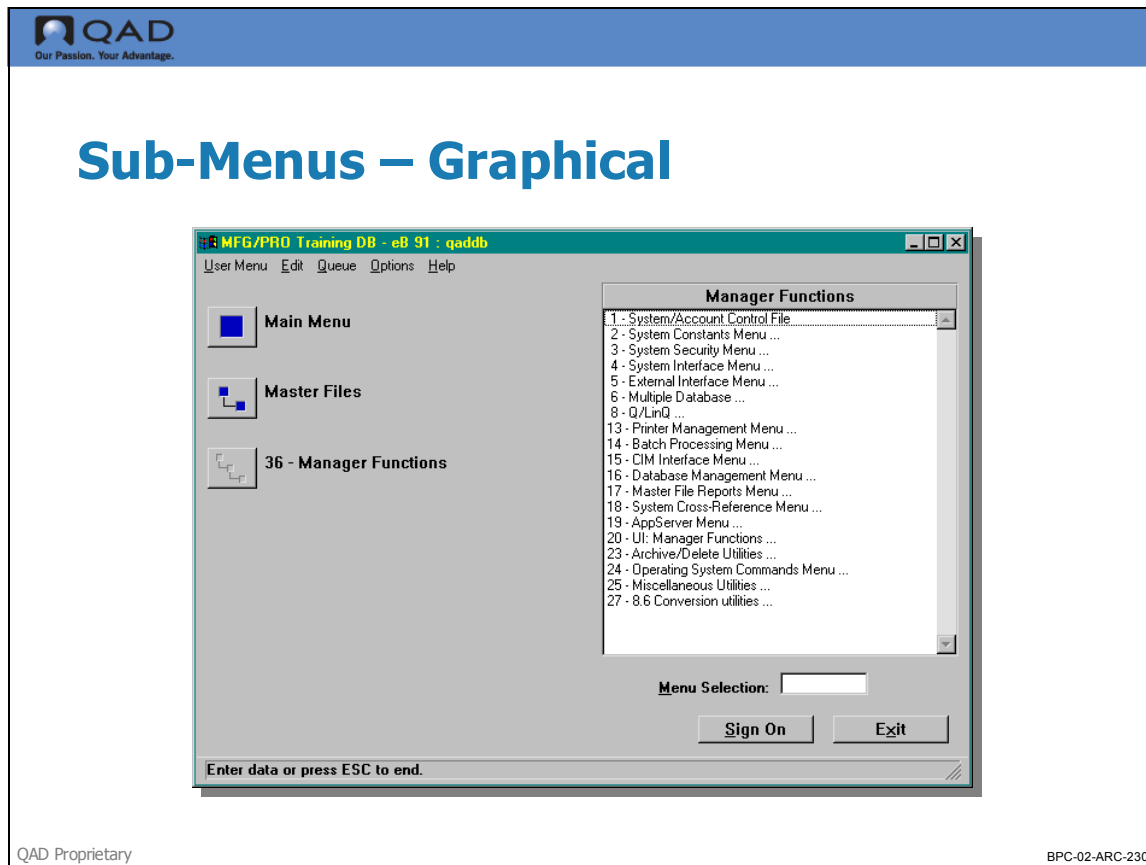
This is primarily intended for UNIX-based environments with character clients.

Main Menu: GUI



This graphical user interface is designed to run on Windows clients in both UNIX and Windows client/server environments.

Sub-Menus: Graphical



Main Menu: QAD Desktop



This browser-based user interface is designed for use over a company intranet or wide-area network. It features screens that display as HTML pages, full text search, multiple detached windows, and a process editor for creating work flows with active URL links.

Menu Execution Options



Menu Execution Options

- ▲ Type full menu number at any prompt (1.4.5)
- ▲ Type program name at menu prompt (apvomt.p)
- ▲ Type partial menu number at a submenu (6.12 at the 36 menu or "." to start from main)
- ▲ Type string from the Name field in Menu System Maintenance at any menu prompt
- ▲ Select program from the User Function Menu

Customizing Menus

QAD
Our Passion. Your Advantage.

Customizing Menus

The screenshot shows the 'Menu System Maintenance' window. On the left, a tree view displays the menu structure under 'Distribution' > 'Purchasing'. 'Purchase Requisition Maintenance' is selected. The right pane shows the configuration for this menu item. The 'Label' field contains 'Purchase Requisition Maintenance'. The 'Image' field has a dropdown menu set to 'Maintenance' and a 'Folder' checkbox. The 'Exec Procedure' field contains 'poprmt.p'. The 'Name' field is empty. The 'Help File' field contains 'poprmt.hlp'. The 'Menu' field contains '5.1' and the 'Selection' field contains '4'.

- ▲ Enter the custom menu number in the "Menu" and "Sel" fields. Enter the new custom program name in the "Exec File" field. This change affects ALL user sessions.
- ▲ The Name field lets you enter a custom string that you can enter in the command line of any QAD Enterprise Applications menu to run the program.

QAD Proprietary
BPC-02-ARC-260

The Name field is good to establish common names for programs. This would be useful when migrating to new versions of QAD Enterprise Applications where the menu numbers have changed.

Upgrading Menus


Our Passion. Your Advantage.

Upgrading Menus

- ▲ Standard Menu numbers may have changed between different QAD Enterprise Applications releases — QAD Enterprise Applications upgrade routines address this
- ▲ To Upgrade custom menu changes:
- ▲ Print Menu System Report in existing DB
 - Print Menu System Report in new DB
 - Use Menu System Maintenance to resolve conflicts with standard menus
 - Use Menu System Maintenance to re-enter custom menus
 - In Desktop, execute the menu build utility before changes can be viewed

QAD ProprietaryBPC-02-ARC-270

Be sure to dump menu definitions in case you can re-use your original custom menu entries (this also goes for messages and help).

Security



Security – Overview

- ▲ Checking a Field for Password Validation
- ▲ Security Types
- ▲ Field Security
- ▲ Customizing Field Security

Field Validations


Our Passion. Your Advantage.

Checking a Field for Validation

Item Data Maintenance X

Go To Actions Copy Print Preview Attach

Attachments

Item Number: 02-0001 Description: CONFIGURED DISPLAY RACK
Unit of Measure: EA

Item Data

Prod Line: 1000

Item Type: CONFIG

Drawing:

Added: 10/31/1990

Status:

Rev:

Design Group:

Group: CARDED

Drawing Loc:

Size:

Promo Group: PG4

Price Break Category:

Control-F (^F) on Field for Field Name & Field Validation Info

QAD Proprietary
BPC-02-ARC-290

In Character and Windows interfaces, you can also access the field on the screen and press ‘CTRL-F’. For eligible fields, the message Password Validation displays. An eligible field must have a specific validation expression in the data dictionary - {gppswd.v &field=<dictionary field name>}.

Security Types



Security Types

- ▲ QAD Enterprise Applications provides options for security on several levels:
 - Database Security
 - Menu Security
 - Field Security
 - General Ledger Account Security
 - Inventory Movement Security
 - Entity Security
 - Site Security
 - Desktop Security
 - Master Table Audits

QAD Proprietary

BPC-02-ARC-300

- Database Security determines whether a user can log into the database. This level of security is always active and requires that user specify a valid user ID and password before they log in.
- Menu Security limits access to menus and menu functions.
- Field Security limits who can update specific fields.
- General Ledger Account Security restricts access to GL accounts.
- Inventory Movement Security enables you to grant or deny group members access to shippers and other transactions using specific movement codes at a site.
- Entity Security limits who can create GL transactions for a particular entity.
- Site Security limits who can create inventory transactions at secured sites.
- Desktop Security lets you restrict who can log into a database using QAD's Desktop interface. It also provides a detailed and focused way of controlling access to data when browses and reports are generated from the Desktop.
- Master Table Audits show when and by whom changes were made to key master tables.

Overview of Security Types

The screenshot shows the 'Standard Transaction Maint' window in QAD. Red arrows point from labels to specific fields: 'Menu Security' points to the window title bar, 'Field Security' points to the 'Effective' date field, 'Account Security' points to the 'Account' field (1040), and 'Entity Security' points to the 'Enty' field (1000).

Menu Security

Field Security

Account Security

Entity Security

Standard Transaction Maint X

Go To Actions Copy Print Preview

Quality Products

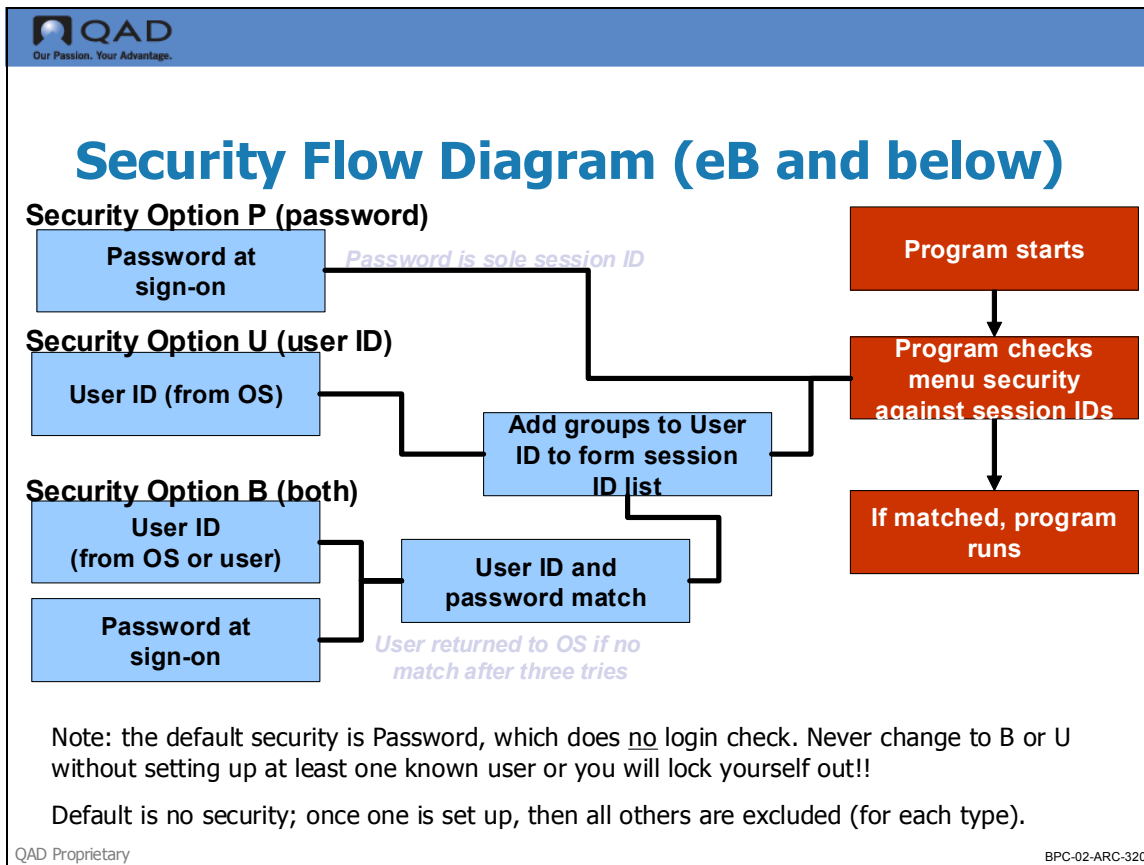
GL Reference: JL081118000001 Type: JL Effective: 11/18/2008 Period: 11/2008
 Currency: USD Control: 1,000.00 Base Total: 0.00
 Daybook: System Entry Number: Correction: ☐

Line	Account	Project	Enty	Description	Cur	Amount
1	1040		1000	Petty Cash	USD	1,000.00

QAD Proprietary BPC-02-ARC-310

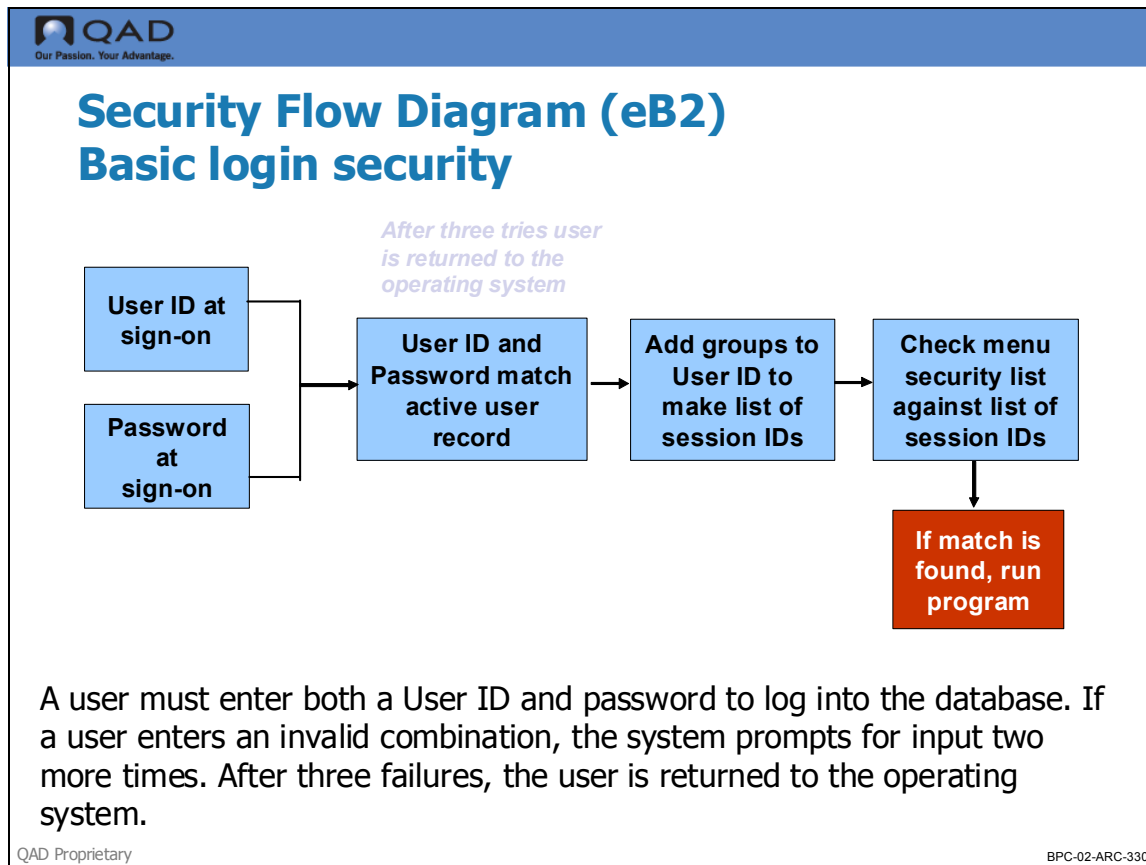
- Menu Security prevents access to screens
- Field Security prevents changing a field
- Entity Security prevents creation of GL transactions for an entity
- Account Security prevents users from updating an account balance
- For Entity and Field Security, specify user IDs
- For Menu Security, Site Security, Inventory Movement Code Security (36.3.17), and GL Account Security, specify any combination of user IDs or user group names

Security Flow Diagram: eB and Below



- P = Prompts for password. Does no validation at login. In QAD Enterprise Applications eB, the security option in Security Control (36.3.24) determines what information a user must supply before logging into a database: User ID, password, or both.
- U = Logs in as OS user and does not prompt for password.
- B = User defaults as OS user. Prompts for user and password.

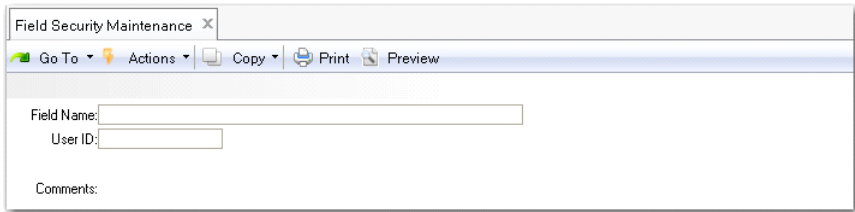
Security Flow Diagram: eB2



Field Security



Field Security (1 of 3)



- ▲ Field security prevents unauthorized users from updating secured fields
- ▲ Using Field Security
 - Activated Field Security Report lists all the fields that have security activated. It also lists privileged user Ids.
 - Dictionary Field Security Report lists fields that can be given security
 - Field Security by Groups adds a group of users to the list of authorized users for a field

QAD Proprietary BPC-02-ARC-340



Field Security (2 of 3)

- ▲ Adding Security to an Eligible Field
 - Add the field name and the list of user IDs that can access the field in Field Security Maintenance
 - Verify that the field is secured by running the Activated Field Security Report
- ▲ Adding Field Security Eligibility
 - You can make most fields eligible for field security by adding the validation expression to the field in the data dictionary
 - Recompile the programs that use the field using the modified data dictionary

Field Security (3 of 3)

- ▲ You can also add field security for a group of users
 - Assign users to groups in User Maintenance
 - Execute Field Security by Group
 - Even with this process, field security is only available at user level, not at group level
 - This is just a batch utility that lets you add multiple individuals simultaneously. This has the following consequences:
 - If you remove a user from a group that was given access to a field, that user can still access the field. To prevent this, use Field Security Maintenance to remove the individual user.
 - You cannot use Field Security by Group to remove a group of users from the list of authorized users. To remove a group, you must remove every individual from the group using Field Security Maintenance.

Custom Field Security



Custom Field Security



- ▲ From the data dictionary, select the field you want to secure

QAD Proprietary

BPC-02-ARC-370



Custom Field Security

Field Properties

Field Name: Member of an Index: no
Data Type: date Member of a View: no

Optional

Format: Examples...
Label:
Column Label: Initial Value:
Order #: Decimals: Description:
Help Text:
☐ Mandatory ☐ Case-Sensitive ☐ Extent 0

Triggers... Validation... View-As... String Attrs... Database...

OK Save Cancel < > Help

▲ Select Field Properties



Our Passion. Your Advantage.

Custom Field Security

Field Validation

Expression that must be TRUE for the entered value:
{gppswd.v &field = "pt_added"}

Field Validation Message:
ERROR: USER DOES NOT HAVE ACCESS TO THIS FIELD. Please re-enter.

OK

Cancel

Help

▲ Select Validation

QAD Proprietary

BPC-02-ARC-390

Password Validation



Password Validation Program for Fields – gppswd.v

```

/* gppswd.v - VALIDATE FIELD PASSWORDS */
/* COPYRIGHT qad.inc. ALL RIGHTS RESERVED. THIS IS AN UNPUBLISHED WORK */
/*F0PN*/ /*V8:convertMode=Maintenance */
/* REVISION: 6.0 LAST MODIFIED: 06/28/90 BY: PML */
/* REVISION: 7.0 LAST Modified: 10/11/91 BY: DGH *D892 */

(not can-find(first flpw_mstr where flpw_field = frame-field and
              flpw_userid = global_userid)
Or {&field} not entered)
#

```

Site Security



Site Security

Site Security Maintenance X

Go To Actions Copy Print Preview Attach

Attachments

Site: 10

User IDs/Groups:
Admin, Sales, Clerks

- ▲ Site Security lets administrators control user access to inventory transactions at each site
- ▲ Access is managed by user and by group. A user can access a site only if that user's ID or group name appears in the Groups field in Site Security Maintenance.

QAD Proprietary

BPC-02-ARC-410

Review



QAD Enterprise Applications Architecture

- ▲ Identify the components of QAD Enterprise Applications
- ▲ Identify the menu structure of QAD Enterprise Applications
- ▲ Identify the security types in QAD Enterprise Applications

Chapter 3

Product Requirements

Introduction



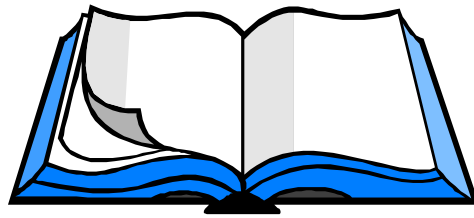
Product Requirements

In this section, you learn how to identify the requirements for customizing QAD Enterprise Applications

- ▲ Progress Database Server
- ▲ Client
- ▲ Development and Administration
- ▲ QAD Enterprise Applications

Terminology

- ▲ Enterprise Database Server
- ▲ Workgroup Database Server
- ▲ Client Networking



Terminology



PROGRESS Products

- ▲ Database server
- ▲ Client and networking
- ▲ Development and administration

Database Server Products

- ▲ Enterprise Database Server
 - Contains RDBMS, server networking, and minimal (Runtime) client product
- ▲ Workgroup Database Server
 - Package is similar to Enterprise Database Server, but without key performance aids such as Before-Image Writers
 - Not recommended for production system



Client Products

▲ Client Networking

- Enables clients to access remote databases running with the proper network parameters
- Runtime client, which executes compiled 4GL programs

▲ Query/Results

- Tool for writing inquiries and reports (only compiles programs that read, not write)
- Required to run QAD Enterprise Applications browses

Development and Admin Products



Development & Admin Products

▲ 4GL

- Available on Windows and UNIX
- Compiles any program, including ones that write to database
- Provides database administration utilities

▲ ProVISION package

- Available on Windows only
- Includes 4GL and other development tools

▲ At least one license of 4GL or ProVISION required for QAD Enterprise Applications installation and administration

QAD Enterprise Applications Product Requirements



QAD Enterprise Applications Product Requirements

- ▲ The QAD Enterprise Applications bundle is a prerequisite for purchasing any other modules:
 - OS Base Module
 - Manufacturing and Distribution base modules
- ▲ Some of the optional modules that are available:
 - Advanced Pricing
 - EDI
 - Payroll
 - Compliance
 - Decision Support
 - Fixed Assets
- ▲ The customer must purchase at least 8 named user or 4 concurrent user licenses
- ▲ Reports and inquiries are also shipped as source

QAD Enterprise Applications Source Code



QAD Enterprise Applications Source Code

- ▲ The following rules apply to source code licensing
 - Use for software development environment only
 - Purchase available on a per module basis
 - Not discountable
 - Requires that a production software license for the same module has already been purchased
 - Software maintenance support is charged at the current list price
 - Includes a right-to-use on a separate CPU for development purposes

QAD Enterprise Applications Available Source Code


Our Passion. Your Advantage.

QAD Enterprise Applications Available Source Code

- Accounts Payable
- Distribution Req. Plan.
- EDI
- Enterprise Ops Planning
- European Accounting
- Fixed Assets
- Forecasting
- Formula/Process
- General Ledger
- Inventory Control
- Master Scheduling
- Material Req. Plan.
- Multiple Currency
- OS module (UNIX or NT)
- Physical Inventory
- Prod. Change Control
- Product Line Planning
- Product Structures
- Pro/Plus
- Purchasing
- Quality Management

- Release Management
(Supplier Schedules)
- Repetitive
- Resource Planning
- Routings/Work Centers
- Sales Analysis
- Sales Orders/Invoicing
- Sales Quotations
- Service/Repair Orders
- Service/Support Mgt.
- Shop Floor Control
- Warehousing API
- Work Orders
- Accounts Receivable
- Advanced Pricing Mgt.
- Capacity Req. Plan.
- Cash Management
- Chinese Accounting
- Compliance
- Configured Products
- Cost Management

QAD Proprietary
BPC-03-PR-090

Review



Product Requirements

- ▲ Identify the requirements for customizing in QAD Enterprise Applications
 - ✓ Progress Database Server
 - ✓ Client
 - ✓ Development and Administration
 - ✓ QAD Enterprise Applications

Chapter 4

PROGRESS Basics

Introduction



PROGRESS Basics

- ▲ In this section, you learn how to identify some Progress basics before customizing in QAD Enterprise Applications

QAD Proprietary

BPC-04-PB-010

While we assume a basic level of understanding of PROGRESS programming conventions, a short review of some relevant subjects is useful. We do this before we include code snippets in this class as a convenience to you. Please consult your PROGRESS manuals for more information on this topic.

File Name Extensions



File Name Extensions

.p	Source Code Files
.i	Include Files
.t	Schema Trigger Files
.w	Source Code Files Created in the UIB
.v	Validation Routines
.r	Executable Files
.st	Multi-Volume Database Structure
.db	Database Extent Directory
.d1/.d2	Database Data Extents
.bi	Database Before-Image File
.lg	Database Log File
.lk	Database Lock File
.d	Data Files
.df	Database Definition Files
.fd	Database Bulkload File

QAD Proprietary

BPC-04-PB-020

PROGRESS dictates the extensions of the files used by QAD Enterprise Applications. Program source code is stored in files with the extensions of .p, .i, .t, .w, and .v. Encryption does not alter the extension of the source file. Compiled source code is stored in files with the extension of .r.

Data files containing data to be loaded into a database have .d extensions. Database definition files have the extension of .df. Descriptions of the database used during bulkloads are .fd files.

PROGRESS requires that each of its databases have one of each of the following files: .db, .bi, .lg. The database is the file with the extension .db. The .bi file contains the “before image” of the database, i.e., the image of the database record before new data is committed. The log file (.lg) contains a record of when the database was opened, closed, etc. The lock file (.lk) exists whenever the database is running.

Comments



Comments

- ▲ Comments in source code are specified as `/* comments */`
- ▲ Anything within markers is not active code

```
if to_part = "" then to_part = hi_char.  
if to_site = "" then to_site = hi_char.  
  
/* SELECT PRINTER */  
{mfse1bpr.i "printer" 80}  
{mfphead.i}
```

Syntax Elements



Syntax Elements

▲ statement	One complete instruction to Progress. A period shows the end of the instruction.
▲ phrase	Unit within a statement that alters or refines processing of the statement
▲ variable	A variable holds data in memory
▲ operator	Language element used to manipulate, compare or test values in an expression
▲ expression	Constant, field or variable name, function, or combination of these and an operator
▲ procedure	Source file containing Progress 4GL code
▲ block	Sequence of statements treated as a unit
▲ function	Performs a discrete task within expressions or statements, usually returns a value

Data Types



Data Types

Data Types	Description
Character	Contains any type of data (letters, numbers, and special characters)
Integer	Holds whole numbers, positive or negative, ranging from -2,147,483,648 to 2,147,483,647
Decimal	Contains decimal numbers up to 50 digits in length, including up to 10 places to the right of the decimal
Date	Holds dates ranging from 1/1/32768 B.C. to 12/31/32767 A.D. (QAD Enterprise Application's hi_date = 12/31/3999)
Logical	Values that evaluate to a Yes/No or True/False

Default Display Formats



Default Display Formats

Format	Description
>	Designates zero suppression: PROGRESS replaces the > character with a number as long as it is not a leading zero. If it is a leading zero, it is replaced with a blank.
,	Displays a comma, unless > precedes it and the character would be a leading zero in which case it is replaced with a null.
.	Displays a decimal point

Data Type	Default Format	Example:	
		Stored Value	Displayed Value
Character	x(8)	Now is the time	Now is t
Integer	->>, >>>, >>9	-12345	-12,345
Decimal	->>, >>9.99	12345	12,345.00
Date	99/99/99	3/10/1993	03/10/93
Logical	yes/no	no	no

Record Retrieval



Record Retrieval

- ▲ FOR [EACH|FIRST|LAST] record-phrase
 - Returns a set of records
 - Looping
 - Block
- ▲ FIND [FIRST|LAST|NEXT|PREVIOUS] WHERE...
 - Returns a single record
 - No block
 - Raises an error if it fails

Scoping



Scoping

- ▲ Record scope controls how long a session “owns” a given record
- ▲ Transaction scope controls how many records are included in a single write to the database
- ▲ Record locking controls how and whether others access records your session is working with

Record Scoping



Record Scoping (1 of 2)

- ▲ Record read from database stored in a record buffer
- ▲ “Scope” is the portion of code during which the record is active
- ▲ Record scope controls:
 - When to write a record to the database
 - When to release a record for other users
 - also depends on type of lock and transaction scope
 - When to validate a record
 - When to reinitialize the index cursor
 - initialized upon entry to procedure block where record is scoped

Record Scoping (2 of 2)

STRONG:	DO FOR <file>: REPEAT FOR <file>:
MEDIUM:	REPEAT: REPEAT WITH <frame>: REPEAT ON ERROR, ENDKEY: FOR EACH:
WEAK:	REPEAT TRANSACTION: FOR FIRST/LAST:
NONE (FREE):	DO: DO TRANSACTION: DO ON ERROR, ENDKEY: FIND:

Transactions



Transactions (1 of 2)

- ▲ A transaction controls:
 - When records are written to the database
 - How long a record lock remains after the data is changed
 - How much data will be written or discarded on completion or cancellation of the transaction
 - Whether variables changed during the transaction will be restored to their original values if a transaction is undone

Transactions (2 of 2)

- ▲ Transactions are scoped to:
 - FOR EACH blocks that update the database
 - REPEAT or REPEAT FOR blocks that update the database
 - DO TRANSACTION or other blocks using the TRANSACTION keyword
 - DO ON ERROR and DO ON ENDKEY blocks
- ▲ If none of these blocks are present and a change to the database is encountered, scope is raised to the entire procedure

Record Locking



Record Locking

▲ NO-LOCK

- Bypasses the normal locking mechanism
 - Allows record reading regardless of lock status
 - Allows record reading without locking records

▲ SHARE-LOCK

- Whenever PROGRESS reads a record (the default lock status), it puts a SHARE-LOCK on that record. Other users can read the record but cannot update it until the procedure releases the SHARE-LOCK.

▲ EXCLUSIVE-LOCK

- Whenever PROGRESS updates a record, it puts an EXCLUSIVE-LOCK on that record. Other users cannot read or update that record until the procedure releases the EXCLUSIVE-LOCK (except NO-LOCK reads).

Frame Scoping



Frame Scoping

- ▲ Frame scope controls the access you have to a frame for the display of records
- ▲ Every statement that displays data or prompts for input uses a frame (or: a frame is scoped to every display statement or block)
- ▲ A frame is scoped to only one block
- ▲ The innermost iterating (FOR EACH, REPEAT) block uses a down frame (displays multiple lines and multiple data values)
- ▲ Default frame for DO blocks is the enclosing FOR EACH, REPEAT, etc., block's frame

Named Frames



Named Frames

- ▲ Four options:
 1. Name the default frame (FOR EACH ... with FRAME a) to reuse the frame in an enclosed block
 - ▲ Takes on default characteristics
 2. Name a new frame for use by the DISPLAY component (DISPLAY ... with FRAME a)
 - ▲ Loses default characteristics (DOWN)
 3. Use the FORM statement
 - ▲ Scopes named frames to the procedure
 - ▲ Allows for extensive definition of appearance and processing
 4. Use the DEFINE FRAME statement
 - ▲ Allows you to scope named frames as needed
 - ▲ Also allows extensive definition

BREAK Group



BREAK Group

- ▲ Used when you want to do some work based on whether the value of a certain field changes over a series of block iterations
- ▲ Use in conjunction with the ACCUMULATE statement and ACCUM function
- ▲ Syntax

```
FOR EACH file WHERE expression BREAK BY field1:  
    IF FIRST-OF(field) THEN...  
    IF LAST-OF(field) THEN...  
END.
```

Include Files



Include Files

- ▲ An include file is a text file that contains PROGRESS source code
- ▲ PROGRESS replaces the reference with the contents of the include file at compile time
- ▲ Include files are placeholders for actual code
- ▲ Since PROGRESS compiles the main procedure and the include file together, they share the same variables and frames
- ▲ Have a .i extension by convention
- ▲ Example
- ▲ {mfdtitle.i}

Internal Procedures



Internal Procedures (1 of 2)

- ▲ PROGRESS allows storage of subprocedures with the main procedure in one procedure file. These subprocedures are called internal procedures.
- ▲ Partial syntax:

```
PROCEDURE PROCEDURE-NAME:  
          STATEMENTS  
END [PROCEDURE].
```

- ▲ By convention, you store procedures at the bottom of a procedure file, after the main procedure

Internal Procedures (2 of 2)

- ▲ Internal procedures do not require a file extension (.p)
- ▲ Access them in exactly the same way as external procedures - with the RUN statement
- ▲ The main procedure cannot use the resources defined in an internal or external subprocedure
- ▲ An internal procedure can use the resources defined in the main procedure

Passing Parameters



Passing Parameters

- ▲ A parameter is a variable in a called procedure whose function is to accept an input value from the calling procedure, output value to the calling procedure, or both

$$\text{DEFINE} \left\{ \begin{array}{l} \text{INPUT PARAMETER} \\ \text{OUTPUT PARAMETER} \\ \text{INPUT-OUTPUT PARAMETER} \\ \text{BUFFER} \end{array} \right\} \text{BufferParamName} \left\{ \begin{array}{l} \text{AS data-type} \\ \text{LIKE field} \\ \text{FOR table} \end{array} \right\}$$

- ▲ Input - a parameter defined in a called procedure that receives a value from the calling procedure
- ▲ Output - a parameter defined in a called procedure that returns a value to the calling procedure
- ▲ Input-output - a parameter defined in the called procedure that can receive a value from the calling procedure and return a value back
- ▲ Buffer - a buffer for a table passed to and from an internal procedure or program

User-Defined Functions



User-Defined Functions

- ▲ Let applications define a logical rule or transformation once, then apply the rule or transformation an unlimited number of times

- ▲ Syntax

```
FUNCTION function-name [RETURNS]  
data-type [(param [, param]...)]  
{FORWARD | [MAP [TO] actual-name] IN  
proc-handle
```

- Returns a single value
- Useful for complex calculations
- No user interface statements
- Local variables allowed

TEMP-TABLES



TEMP-TABLES

- ▲ Defines a temporary table. PROGRESS stores temporary tables on disk in a temporary database.

- ▲ Syntax

```
DEFINE [[NEW [GLOBAL]] SHARED] TEMP-TABLE
  temp-table-name [NO-UNDO] [FIELD field-name
    {AS data-type} [field-options]]... [INDEX
    index-name [IS [UNIQUE] [PRIMARY] ]
    {index-field [ASCENDING | DESCENDING]}...]...
```

- Use of indexes allow statements that require indexes such as FOR EACH or a simple FIND
- Lasts only for the duration of the procedure (or session if defined globally) that defines them and only one user at a time can access them

- ▲ Use EMPTY TEMP_TABLE statement to empty a temp-table in order to reuse it

- ▲ Syntax

```
EMPTY TEMP-TABLE <table-name> [NO-ERROR]
```

Buffers



Buffers

- ▲ PROGRESS provides one buffer for each table in a procedure
- ▲ PROGRESS uses that buffer to store one record at a time from the table as records are needed during the procedure
- ▲ If more than one record is needed at a time from a table, use the DEFINE BUFFER statement to define additional buffers for that table
- ▲ Syntax

```
DEFINE [ [NEW] SHARED ] BUFFER buffer FOR table  
[PRESELECT] [LABEL label]
```

Preprocessor Directives



Preprocessor Directives

- ▲ Method for naming a local or global constant (text or value) that you may reuse often in your procedure or stack
- ▲ &SCOPED-DEFINE
 - Sets a value to the end of the defining file
- ▲ &GLOBAL-DEFINE
 - Sets value to end of compilation unit
- ▲ Evaluate using &IF &THEN &ELSE &ENDIF statements

Review



PROGRESS Basics

- ✓ Identify some Progress basics before customizing in QAD Enterprise Applications

Chapter 5

Functionality Reuse

Introduction



Functionality Reuse

- ▲ In this section, you learn how to identify programs in QAD Enterprise Applications that can be re-used for customizing
 - Printing
 - Batch submittal
 - Screen/Report headers and trailers
 - Scrolling look-up windows
 - E-mail

QAD Proprietary

BPC-05-RE-010

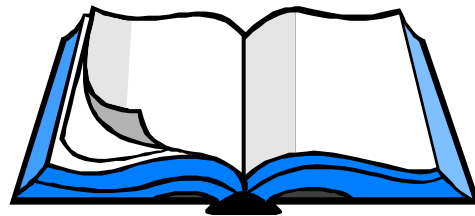
QAD has developed an extensive programming infrastructure that is available for you to reuse in your programming of QAD Enterprise Applications customizations. Printing, batch submittal, report headers and trailers, scrolling help windows, to name a few.

Terminology



Terminology

- ▲ Production environment
- ▲ Development environment
- ▲ Batch processing
- ▲ Triggers



QAD Proprietary

BPC-05-RE-020

Production Environment

This is the working environment where the business' day-to-day business transactions are entered.

Development Environment.

This is the environment used by developers where the program path includes files that are currently in the development directory, test directory, stage directory, or release directory, in that order.

Batch Processing

Batch processing is a program that is assigned to the computer to run without further user interaction. Batch jobs are usually initiated by a system user. Some are defined to run automatically at a certain time.

Triggers

A piece of code that Progress executes on behalf of an application when a certain database action occurs or a certain user interface event occurs.

Common Include Files



Common Include Files

▲ mfdtitle.i	Screen Title, Revision, & Date
▲ mfdeclre.i	Global Variables
▲ gprun.i	Subroutine Execution
▲ mfnfp.i	Next/Previous Function
▲ mfmsg.i	Message Display
▲ mfmsg01.i	Message Display with Update
▲ mfselprt.i	Select Printer
▲ mfselbpr.i	Select Printer w/ Batch output

Common Include Files

- ▲ mfrpchk.i Report End
- ▲ mfredet.i Printer Reset
- ▲ mfquoter.i Primes data for batch mode
- ▲ mfphead.i Report Header
- ▲ mfrtrail.i Report Trailer
- ▲ mfnctrl.i Retrieve the next control file number
- ▲ mfnctrlc.i Retrieve the next control file number with a prefix

eB Common Include Files



eB Common Include Files

- ▲ pxsevcon.i API Preprocessor Defines
- ▲ pxmaint.i Standard Components
- ▲ pxfunct.i Dynamic Function Calls
- ▲ pxrun.i All Purpose Run Interface
- ▲ pxmsg.i All Purpose Message Interface
- ▲ gpselout.I Select output destination
- ▲ gpdirpre.i Isolates OS dependency for pathnames to a single location
- ▲ pxpgmmgr.i Retrieving program manager for persistent procedures

QAD Proprietary
BPC-05-RE-050

pxsevcon.i

Holds preprocessor definitions necessary for using pxmsg.I and pxrun.i

pxmaint.i Includes pxsevcon.i

Holds standard data conversion functions: boolToChar, charToDate, charToBool, charToInt, charToDecimal, charToInt, dateToChar, decimalToChar, intToChar. Holds generic function: stripDirFromProgName, getHandle

pxfunct.i

Parameters: &FUNCTION, &PROGRAM, &HANDLE, &PARAM, &PARENT

pxrun.i

Parameters: &PROC, &PROGRAM, &HANDLE, &PARAM, &NOAPPERROR, &CATCHERROR, &FIELD-LIST, &MESSAGE-LIST, &PARENT

pxmsg.i

Requires pxsevcon.I minimally, or pxmaint.I.

Possible parameters: &MSGNUM, &MSGTEXT, &ERRORLEVEL, &CONFIRM,
&MSGARG[1...5], &MSGBUFFER, &CONFIRM, &CONFIRM-TYPE, &BUTTONDATA,
&CALLINGPGM, &FIELDNAME, &PAUSEAFTER, &CONTEXT

mfdtitle.i



mfdtitle.i (1 of 2)

- ▲ mfdtitle.i displays the title line at the top of the screen which includes
 - Program name
 - Revision level
 - Menu selection
 - Date
- ▲ Must be included in every main program
- ▲ The revision level is passed as a parameter to mfdtitle.i

mfdtitle.i (2 of 2)

- ▲ The revision level of a custom program should be increased every time a change is made to that custom program
- ▲ The {mfdtitle.i} include file incorporates {mfdeclre.i} as well



mfdtitle.i - how to use it...

```

Procedure Editor - C:\Documents and Settings\rkl\My Documents\Info...
File Edit Search Buffer Compile Tools Options Help

/* adcsrp.p - CUSTOMER REPORT */
/* Copyright 1986-2006 QAD Inc., Carpinteria, CA, USA. */
/* All rights reserved worldwide. This is an unpublished work. */
/* $Revision: 1.4.1.9 $ */
/*V8:ConvertMode=FullGUIReport */
/* REVISION: 4.0 LAST MODIFIED: 02/29/88 BY: FML *A113 */
/* REVISION: 7.0 LAST MODIFIED: 06/03/94 BY: dpm *FO58 */
/* REVISION: 8.6 LAST MODIFIED: 10/13/97 BY: GYK *KOTZ */
/* REVISION: 8.6 LAST MODIFIED: 10/29/97 BY: *H1G4* Manish K. */
/* REVISION: 9.0 LAST MODIFIED: 03/10/99 BY: *MOB3* Michael Amaladhas */
/* REVISION: 9.0 LAST MODIFIED: 03/13/99 BY: *MOBD* Alfred Tan */
/* REVISION: 9.1 LAST MODIFIED: 03/24/00 BY: *NOST* Annasaheb Rahane */
/* REVISION: 9.1 LAST MODIFIED: 08/11/00 BY: *NOKK* jyn */
/* Old ECO marker removed, but no ECO header exists *FOPN */
/* Revision: 1.4.1.7 BY: Katie Hilbert DATE: 04/01/01 ECO: *P002 */
/* $Revision: 1.4.1.9 $ BY: Paul Donnelly (SB) DATE: 06/26/03 ECO: *Q00B */
/*--Revision end-----*/

/*
*****
/* All patch markers and commented out code have been removed from the source */
/* code below. For all future modifications to this file, any code which is */
/* no longer required should be deleted and no in-line patch markers should */
/* be added. The ECO marker should only be included in the Revision History. */
*****
*/

(mfdtitle.i "2+ ")
define variable code like cm_addr.
define variable code1 like cm_addr

```

QAD Proprietary

BPC-05-RE-080



Our Passion. Your Advantage.

mfdtitle.i - what it does...

Customer Address Report

Go To

Actions

Copy

Print

Preview

Customer:

Name: ABLE Engineering Pty Ltd

Postal:

To:

To: Worldwide Supply

To:

Output:

page

Batch ID:

Customer Address Report (2.1.3)

QAD Proprietary

BPC-05-RE-090





mfdtitle.i - where it gets info...

The screenshot shows the 'Menu System Maintenance' window. The left pane displays a tree view of the menu structure under 'Addresses / Taxes > Customers Menu'. The 'Customer Address Report' item is selected. The right pane shows the configuration for this item:

- Type: .NET UI Menu
- Language: us
- Label: Customer Address Report
- Image: Report (dropdown menu)
- Folder: ☐
- Exec Procedure: adcsrp.p
- Name:
- Help File: adcsrp.hlp
- Menu: 2.1
- Selection: 3

Menu System Maintenance (36.4.4)

mfdeclre.i



mfdeclre.i (1 of 2)

- ▲ mfdeclre.i defines global variables
- ▲ The parameter to pass to mfdeclre.i is one of the following:
 - &NOSHARED → to suppress the shared variables
 - NEW → to define new shared variables
 - blank → to define shared variables
 - new global → the only time the parameter should be passed as "new global" is in mfl.p or in a stand-alone procedure



mfdeclre.i (2 of 2)

- ▲ Just as mfdtitle.i must be included in all main programs, mfdeclre.i should be included in all subprograms
 - mfdeclre.i need not be included in a main program because it is included within mfdtitle.i itself
 - Can replace with mfglobal.i for a smaller subset of the global variable definition contained in mfdeclre.i

Exercise 1



Exercise 1 (20 minutes)

- ▲ Start a character QAD Enterprise Applications session using the Reflection Icon on your desktop and go to the Progress Editor
- ▲ Look at the File Relationship Diagram of Sales Orders and determine the relationship between Items, Sales Order Detail Lines and Sales Order Master
- ▲ Create the following query

```

R,.....T
.      From: 1      To:      .
F,.....G
R,.....T
. Item Number: 22-100      Snoer UK      Prod Line: 1000.
F,.....G
R,.....T
. Order Sold-To Ln Qty Ordered Qty to Invoice.
. ....
. S9900003 cust3 1      8,0      0,0.
.
    
```

Global Variables



Global Variables

mfdeclre.i

{1} "NEW GLOBAL" can be specified as the argument if this is to be included in mf1.p and in other stand-alone programs.
{1} "" if this is included in subprograms requiring variables

define {1} shared variable global_part like pt_part

sosomt.p

{mfddtitle.i "f"}

sosomta.p

{mfdeclre.i}

gprun.i



gprun.i – Run a Program

- ▲ gprun.i should be used to run all subprograms
- ▲ It uses the global user language directory variable (defined in mfdeclre.i) to ensure that the program to be executed is run from the correct language and module subdirectory
- ▲ gprun.i parameters
 - {1} = program name
 - {2} = parameters to pass to the program
 - {3} = optional [PERSISTENT [set handle]] parameter for Progress V7
- ▲ Example in sosomt.p:


```
{gprun.i ""sosomt1.p"" "(input no)"}
```

pxrun.i



pxrun.i – Run a Program or Procedure (1 of 2)

- ▲ The 'pxrun.i' construct is used to execute internal procedures instead of the Progress 'run' statement or 'gprunp.i'
- ▲ Runs both programs and procedures
- ▲ Runs in both normal and API mode
- ▲ pxrun.i has a robust list of parameters:
 - &PROC, &PROGRAM, &HANDLE, &PARAM,
 - &NOAPPERROR, &CATCHERROR, &FIELD-LIST,
 - &MESSAGE-LIST, &PARENT

QAD Proprietary

BPC-05-RE-166

It is important for the procedure name and program name to be on the same line for grepping purposes.



pxrun.i – Run a Program or Procedure (2 of 2)

▲ Example in pomt.p:

```
{pxrun.i &PROC='processRead'  
  &PROGRAM='popoxr.p'  
    &PARAM="(input ponbr,  
      buffer po_mstr,  
      input yes,  
      input yes)"  
    &NOAPPERROR=TRUE  
  &CATCHERROR=TRUE}
```



Subroutine Execution

Production Environment

PROPATH = : / qad

{gprun.i}

```
run value( global_user_lang_dir +
  substr({1},1,2) + '/' + {1} ) {3}
  {2}.
```

Development Environment

PROPATH = : / qad / src

{gprun.i}

```
run value ( {1} ) {3} {2}.
```

When creating a development environment, it might be desirable to run from source code rather than compiled code. As the standard version of gprun.i will look in the appropriate two-letter subdirectory for a compiled version of a procedure, modifying standard gprun.i as shown in the example will make it possible to run from source code only in a development environment.

QAD Proprietary

BPC-05-RE-180

For development environments, you not only edit gprun, but also gpwinrun.p, gprun1.i and mfdecgui.

mfnp.i**mfnp.i – Next/Previous Function (1 of 2)**

- ▲ Enables a user to use arrow keys to scroll through records on a key field
- ▲ Can only be used in editing blocks



mfnp.i – Next/Previous Function (2 of 2)

▲ Example

```
{mfdeclre.i}
prompt-for pt_part
editing:
    if frame-field = "pt_part" then do:
        /* FIND NEXT/PREVIOUS RECORD */
        {mfnp.i pt_mstr pt_part pt_part pt_part pt_part
        pt_part}
        if recno <> ? then
            display pt_part pt_desc1.
        end.
    else do:
        readkey.
        apply lastkey.
    end.
end.
```

▲ There are variants of this program that allow for different types of search criteria

Next/Previous Function

mfnp.i

{1} File name
{2} First Input Variable
{3} First Field
{4} Second Input Variable
{5} Second Field
{6} Index

mfnp.i

**Searches through {1} where {3} > input {2}
and {5} >= input {4}
use-index {6} no-lock no-error.**

mfmsg.i



mfmsg.i – Display a Message (below 9.0)

- ▲ mfmsg.i displays a message from the msg_mstr table
- ▲ mfmsg.i requires two parameters:
 - message number
 - error level
 - mfmsg01.i requires a third parameter, a variable to update
 - This include file is used when confirming a delete or verifying the correctness of information
- ▲ For consistency, accuracy, translations and re-use, mfmsg.i should always be used
- ▲ Messages should NEVER be hard coded
- ▲ Message numbers greater than 9000 have been reserved for custom use

Message Display

`{mfmsg.i {1} {2} }`

`{1}` msg number

`{2}` 1-normal 2-warning 3-error 4-error w/o "Please re-enter"

`{mfmsg01.i {1} {2} {3} }`

`{1}` msg number

`{2}` 1-normal 2-warning 3-error

`{3}` boolean variable to update

If you use a mfmsg* program in GUI and it's not a batch process, the message will appear in an alert-box.



Message Display – Include Files (1 of 2)

- ▲ **mfmsg.i** (message number, severity)

```
{mfmsg.i 1 1} /* Adding new record */
```
- ▲ **mfmsg01.i** (message number, severity, variable to update)

```
{mfmsg01.i 11 1 del-yn} /* Please confirm delete */
```
- ▲ **mfmsg01c.i** (message number, severity, character variable to update)
 - Messaging include file for updating all different variable types. This include file does not exist in the character environment.



Message Display – Include Files (2 of 2)

- ▲ **mfmsg02.i** (message number, severity, variable to append onto the end of the message)

- Messaging include file for concatenated variables

```
Error: DELETE NOT ALLOWED, PURCHASE ORDER
      EXISTS PO232.
```

- ▲ **mfmsg03.i** (message number, severity, optional value 1, optional value 2, optional value 3)

```
{mfmsg03.i 1263 3 addr2 site1 """}
/* Schedule SO already exists for ship-
to #, site # */
```

QAD Proprietary

BPC-05-RE-250

mfmsg03.i

This routine substitutes a “#” in the msg_desc with the parameter passed to the routine. Up to five parameters can be passed. Variables msg_var1 and msg_var2 will have to be passed to ensure that two parameters are always passed.

pxmsg.i



pxmsg.i – Display a Message (eB and up) (1 of 2)

- ▲ Always use pxmsg.i to invoke a message in QAD Enterprise Applications for eB and above. Do not use mfmsg*.i procedures in release eB and above.
- ▲ In eB, the mfmsg*.i procedures were consolidated into a single procedure named pxmsg.i.
- ▲ pxsevcon.i defines the &ERRORLEVEL options that can be used for message handling. It must be included in the procedure file where pxmsg.i is included to access the &ERRORLEVEL options instead of the hard-coded numeric error level values (1, 2, 3, 4).

QAD Proprietary
BPC-05-RE-286

pxmsg.i requires pxsevcon.i minimally, or pxmaint.i.

Possible parameters:

- **&MSGNUM** - (optional) The message number for the message in the msg_mstr file. Either &MSGNUM or &MSGTEXT must be specified.
- **&MSGTEXT** - (opt) Additional text to add to the end of the message.
- **&ERRORLEVEL** – (mandatory) The error level (severity) of the error.
 - 1 = normal
 - 2 = warning
 - 3 = error w/ “Please re-enter.”
 - 4 = error w/o “Please re-enter.”

If not specified, the default value 1 will be used.

- **&MSGARG[1...5]** - (opt) An argument that will be substituted into the message text where a “#” character is found, or if no “#” is found, the text is appended to the end. If not specified, the default value “” will be used.
- **&MSGBUFFER** - (opt) The name of a variable where the message text should be placed. If this parameter is specified, the message will not be displayed.

- **&CONFIRM** - (opt) Indicates this is a confirmation message. The name of a variable of type logical which will be updated with the response to the confirmation. Either &CONFIRM and &MSGBUFFER can be specified, but not both.
- **&CONFIRM-TYPE** - (opt) Indicates the confirmation field's type.
- **&BUTTONDATA** - (opt) If this is a confirmation message, then this parameter must be defined to specify the button data associated for the confirmation dialog.
- **&CALLINGPGM** - (opt) Indicates the name of the calling program making the request. If this parameter is not supplied, the name of the program including this include file will be used.
- **&FIELDNAME** - (opt) Displays the field on which the error occurred in JavaUI mode. The field is clickable and will move the cursor to that field.
- **&PAUSEAFTER** - (opt) Set to TRUE if a pause is required after the message. The b batchRun code is included and works appropriately for API mode. Can pause or a time setting can be used.
- **&CONTEXT** - (opt) Sometimes needed in API mode; defines current context.



pxmsg.i – Display a Message (eB and up) (2 of 2)

- ▲ pxmsg.i has many parameters: &MSGNUM, &MSGTEXT, &ERRORLEVEL, &CONFIRM, &MSGARG[1...5], &MSGBUFFER, &CONFIRM, &CONFIRM-TYPE, &BUTTONDATA, &CALLINGPGM, &FIELDNAME, &PAUSEAFTER, &CONTEXT.

pxmsg.i – Display a Message (eB and up)

▲ Example in pjprjxr1.p

```
/* MESSAGE #8785 - ACTIVITY NOT ALLOWED FOR  
SUBPROJECT */  
/* WITH STATUS # */  
{pxmsg.i  
  &MSGNUM=8785  
  &ERRORLEVEL={&APP-ERROR-RESULT}  
  &MSGARG1=projectLine.pjd status  
  &FIELDNAME='projectLineId'  
  return error {&APP-ERROR-RESULT}.
```

gpselout.i



gpselout.i – Printer Selection (1 of 5)

- ▲ QAD Enterprise Applications eB and above: The include file gpselout.i should be used to select an output destination. It should be used with mfredet.i.
 - Calls to mfredetbp2.i, mfredetp02.i, mfredetp03.i and mfredetpr1.i have been replaced by calls to gpselout.i
 - Calls to mfredetprt.i and mfredetbpr.i have been retained, but these include files make a pass-through call to gpselout.i. As programs that call these two include files are updated, these calls should be replaced with direct calls to gpselout.i.

gpselout.i – Printer Selection (2 of 5)

- Enable selection of output destination and batch ID
 - Allow user to select output device and batch ID
 - Validate output device and batch ID
- Redirect output to the destination selected
 - Configure destination if it is configurable
 - Set a stream to output to that destination



gpselout.i (3 of 5)

Example: Here is an example where a call to mfselbpr.i is replaced by a call to gpselout.i.

```
/* {mfselbpr.i printer 132 page "stream repout"} */  
/*CALL GENERIC OUTPUT DESTINATION SELECTION INCLUDE.*/  
{gpselout.i  
    &printType = "printer"  
    &printWidth = 132  
    &pagedFlag = "page"  
    &stream = "stream repout"  
    &appendToFile = " "  
    &streamedOutputToFile = " "  
    &withBatchOption = "yes"  
    &displayStatementType = 1  
    &withCancelMessage = "yes"  
    &pageBottomMargin = 6  
    &withEmail = "yes"  
    &withWinprint = "yes"  
    &defineVariables = "yes"}
```

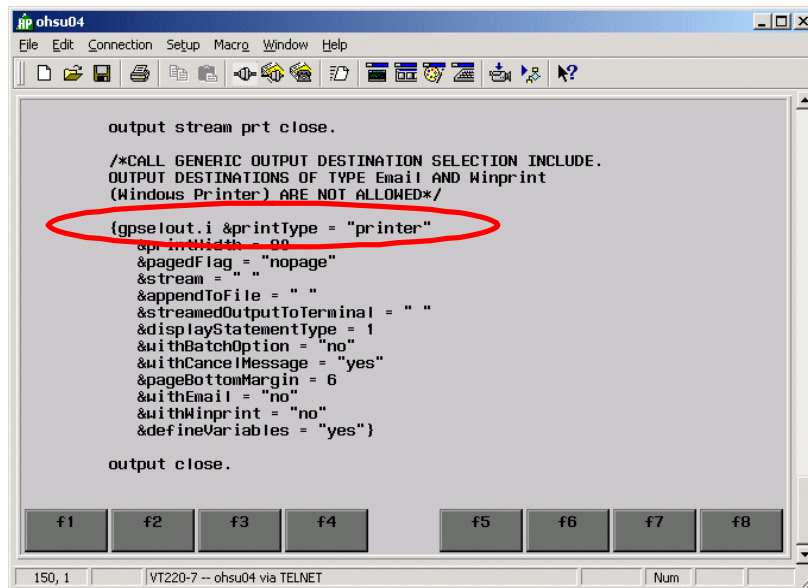
gpselout.I (4 of 5)

```

/*ACCEPT PRINTER INFORMATION*/
/* {mfselbpr.i "printer" "80"} */
/*CALL GENERIC DESTINATION SELECTION INCLUDE SPECIFYING THAT THE
VARIABLES batch_id AND dev SHOULD NOT BE DEFINED AS THEY HAVE
ALREADY BEEN DEFINED WITH A CALL TO mfsprtdf.i AT THE BEGINNING OF
THIS
FILE. */
{gpselout.i &printType = "printer"
            &printWidth = 80
            &pagedFlag = " "
            &stream = " "
            &appendToFile = " "
            &streamedOutputToFile = " "
            &withBatchOption = "yes"
            &displayStatementType = 1
            &withCancelMessage = "yes"
            &pageBottomMargin = 6
            &withEmail = "yes"
            &withWinprint = "yes"
            &defineVariables = "yes"}
.....report code .....
{mfreset.i}

```

gpselout.I (5 of 5)



```
output stream prt close.  
  
/*CALL GENERIC OUTPUT DESTINATION SELECTION INCLUDE.  
OUTPUT DESTINATIONS OF TYPE Email AND Winprint  
(Windows Printer) ARE NOT ALLOWED*/  
{gpselout.i &printType = "printer"  
  &printWidth = 80  
  &pagedFlag = "nopage"  
  &stream = " "  
  &appendToFile = " "  
  &streamedOutputToTerminal = " "  
  &displayStatementType = 1  
  &withBatchOption = "no"  
  &withCancelMessage = "yes"  
  &pageBottomMargin = 6  
  &withEmail = "no"  
  &withWinprint = "no"  
  &defineVariables = "yes"}  
  
output close.
```


mfselprt.i

mfselprt.i – Printer Selection

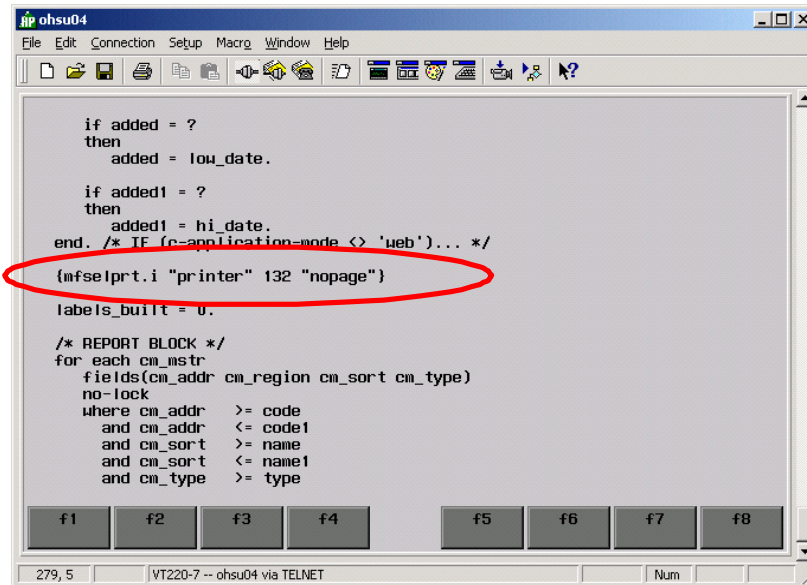
- ▲ mfselprt.i is called the printer selection include file, but it is actually used for output selection

mfselprt.i

{1} default device type ("terminal" or "printer")
{2} width (80 or 132)
Optional parameters
{3} "nopage" for printing checks or labels
{4} stream name (also included in mfrtrail.i)
{5} "append" for disk files

If the third parameter "nopage" is omitted, the default page length is 66 or the value of prd_length, if available. If parameter {3} is set to nopage, the page length is zero.

mfselfprt.i – how to use it...

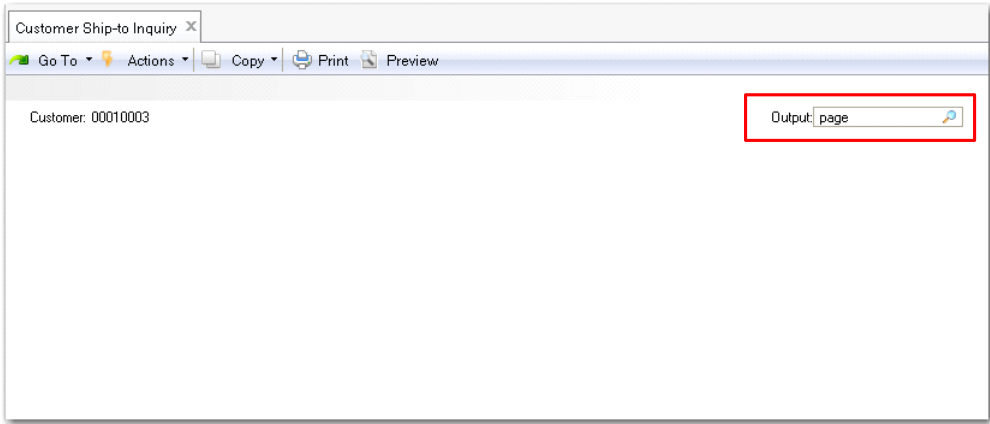


```
if added = ?
then
  added = low_date.

if added1 = ?
then
  added1 = hi_date.
end. /* IF (c-application-mode <> 'ueb')... */
{mfselfprt.i "printer" 132 "nopage"}
labels_built = 0.

/* REPORT BLOCK */
for each cm_mstr
  fields(cm_addr cm_region cm_sort cm_type)
  no-lock
  where cm_addr >= code
    and cm_addr <= code1
    and cm_sort >= name
    and cm_sort <= name1
    and cm_type >= type
```

mfselfprt.i – what it does...



Customer Ship-to Inquiry (2.1.14)


Our Passion. Your Advantage.

mfselfprt.i – where it gets info...

Printer Setup Maintenance

Go To Actions Copy Print Preview

Printer Definition

Output To: winPrint

Description: hewlett packard 2934

Max Pages: 0

Device Pathname: ip -dohlp12 -s

Destination Type: Default

Printer Type: HP2934

Lines / page: 66

Scroll Output: ☐

Spooler: ☒

Printer Control

Initialize Command:

Initialize Ctrl:

80 Column Start: /027&k05

132 Column Start: /027&k25/013

Reset Ctrl: /027&k05

Reset Command:

Delete

Back

Next

Printer Setup Maintenance (36.13.2)

QAD ProprietaryBPC-05-RE-370

mfselbpr.i



mfselbpr.i – Batch Option

- ▲ If a batch option is required when selecting output, incorporate the mfselbpr.i include file into the code rather than the mfselprt.i
- ▲ The batch option displays the batch ID field on the screen below the printer field and allows the user to enter a Batch ID
- ▲ The parameters for mfselbpr.i are identical to those for mfselprt.i



mfselbpr.i – how to use...

```

Open {
  {mfquoter.i name1 }
  {mfquoter.i zip }
  {mfquoter.i zip1 }

  if code1 = "" then code1 = hi_char.
  if name1 = "" then name1 = hi_char.
  if zip1 = "" then zip1 = hi_char.

end.
/* Select printer */
{mfselbpr.i "printer" 132}
{mfthead.i}

for each cm_mstr where (cm_addr >= code
  and cm_addr <= code1)
  and (cm_sort >= name
  and cm_sort <= name1)
  no-lock by cm_sort with frame b width 132 no-box no-attr-space:

  find ad_mstr where ad_addr = cm_addr no-lock.
  if (ad_zip >= zip and ad_zip <= zip1) then do:
    if page-size - line-counter < 3 then page.

```

mfselbpr.i – what it does...

Customer Address Report

Go To

Actions

Copy

Print

Preview

Customer:	To:
Name: ABLE Engineering Pty Ltd	To: Worldwide Supply
Postal:	To:

Output: page

Batch ID: daily

Customer Address Report (2.1.3)

QAD
Our Passion. Your Advantage.

mfselbpr.i – what it does...

Batch Request Detail Maintena... X

Go To Actions Copy Print Preview

Domain: QP
Batch ID: train
Submit Date: 11/19/2008
Submit Time: 08:42:07
Submitted By: mfg

Program:
Run Date:
Run Time:
Parameter Data:

Priority:
Permanent: ☐
Process: ☐
Output:

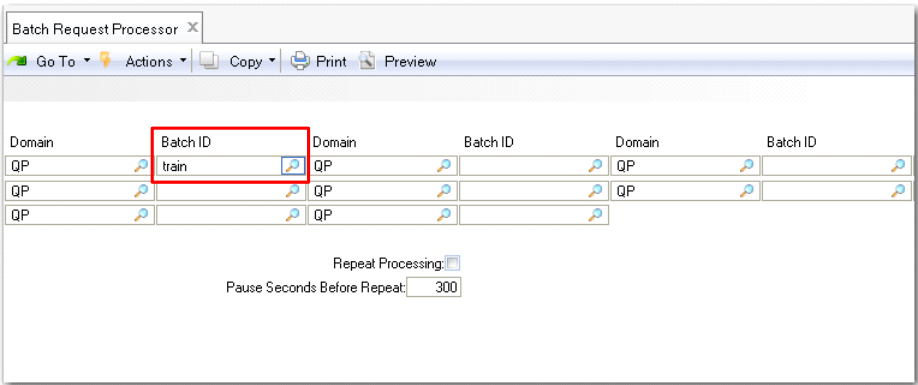
Run Status:

Back Next

Batch Request Detail Maintenance (36.14.3)

QAD Proprietary BPC-05-RE-410

mfselbpr.i – what it does...



Batch Request Processor (36.14.13)

mfrpchk.i**mfrpchk.i – Report End**

- ▲ The mfrpchk.i include file enables the user to terminate a report before reaching the end of the data
- ▲ In addition, the report will terminate if the value of global variable maxpage is exceeded (set in printer maintenance)
- ▲ All parameters passed to mfrpchk.i are optional

mfrpchk.i

{&label} optional label of block to exit.
{&warn} false if report terminated message not to be printed.
Default is to print warning.
{&stream} optional name of output stream, e.g. "(prt)".

mfreset.i



mfreset.i – Printer Reset

- ▲ The function of the mfreset.i include file is to reset the printer and close the output stream
- ▲ There is one optional parameter for mfreset.i, a stream name

mfreset.i

{1} "stream ..." if necessary
Displays scrolled output - prd_scroll_only
Closes output stream
Executes printer reset command if selected - prd_rset_pro

mfquoter.i



mfquoter.i – Report Batch Processing

The purpose of the include file mfquoter.i is to insert double quotes around the selection criteria for reports. By doing so, it allows reports to be run in batch mode.

As mfquoter.i has only one parameter, multiple mfquoter.i statements are required, one for each selection criteria. Notice that mfquoter.i is included after the statements updating the form's selection criteria.

ppptrp.p

```
form
  line    colon 15
  line1   label {t001.i} colon 49 skip
  part    colon 15
  part1   label {t001.i} colon 49 skip

  update line line1 part part1
  {mfquoter.i line }
  {mfquoter.i line1 }
  {mfquoter.i part }
  ...
```

mfphead.i and mfphead2.i



mfphead.i and mfphead2.i

- ▲ mfphead.i and mfphead2.i print report headings
 - name of the procedure that generates the report
 - revision level of the procedure
 - menu option, date, page number, report title & time
- ▲ {mfphead.i "stream rport" "(rport)"}
 - {1} "STREAM stream-name"
 - optional - but if used must include STREAM keyword
 - {2} "(stream-name)"
 - optional - needed if {1} is used (omit keyword)



Report Header

adcsbkrp f	2.21.3 Customer/Supplier Bank Report	Date: 05/22/03
Page: 1	Public Progress Test 92	Time: 10:07:17

- ▲ Each report print-out has a header
 - mfphead.i - 132 column report output
 - mfphead2.i - 80 column report output

mfphead.i



mfphead.i – how to use it...

 **xterm**

```

{mfquoter,i zip      }
{mfquoter,i zip1     }

if code1 = "" then code1 = hi_char,
if name1 = "" then name1 = hi_char,
if zip1 = "" then zip1 = hi_char,

/* Select printer */
{mfselbpr,i "printer" 132}

{mfphead,i}

for each cm_mstr where (cm_addr >= code
and cm_addr <= code1)
and (cm_sort >= name
and cm_sort <= name1)
no-lock by cm_sort with frame b width 132 no-box no-attrib-space:

    find ad_mstr where ad_addr = cm_addr no-lock,
    if (ad_zip >= zip and ad_zip <= zip1) then do:
    if page-size - line-counter < 3 then page,
    display cm_addr ad_name format "x(27)"

```

QAD Proprietary
BPC-05-RE-486

QAD
Our Passion. Your Advantage.

mfphead.i – where it gets info...

Company Address Maintenance

Go To Actions Copy Print Preview

Company Address

Address: ~Reports

Name: United States

Address: Test Database 92 Progress

Address:

Address:

City: State: Post: Format: 0

Country: United States of America usa County:

Attention:

Telephone: Ext: Tel: Ext:

Fax/Telex: Fax/Telex:

Tax Report: Name Control: Last Filing:

Delete Back Next

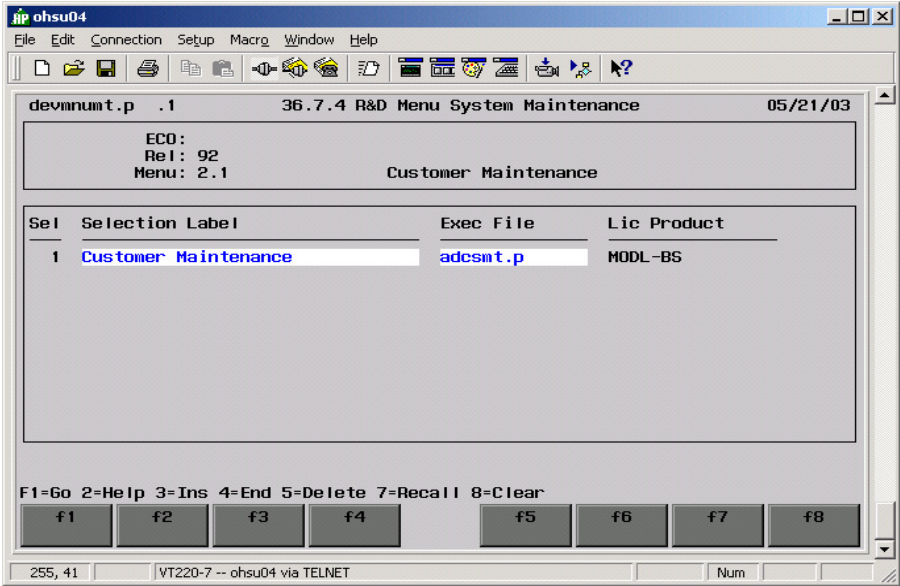
Company Address Maintenance (2.12)

QAD Proprietary BPC-05-RE-490



Our Passion. Your Advantage.

mfphead.i – where it gets info...



QAD Proprietary

BPC-05-RE-500

mfrtrail.i**mfrtrail.i – Report Trailer**

- ▲ Prints the report trailer
- ▲ The printer reset routine mfreset.i is run when either mfrtrail.i or mftrl080.i is executed
- ▲ Custom reports should use the mfrtrail.i and mftrl080.i include files to produce reports that are consistent with those produced by standard QAD Enterprise Applications
- ▲ {mfrtrail.i "stream rport"}
 - {1} "stream name" (optional)

Report Trailers

REPORT CRITERIA:

Report submitted by: pfh

Address: 1000

To: 2000

Sort Name:

To:

Post:

To:

Output: adcsrkrp

Batch ID:

mfrtrail.i

displays 132 column report criteria

{mfreaset.i {1}}

{mfmsg.i 9 1}

mftrl080.i

displays 80 column report criteria

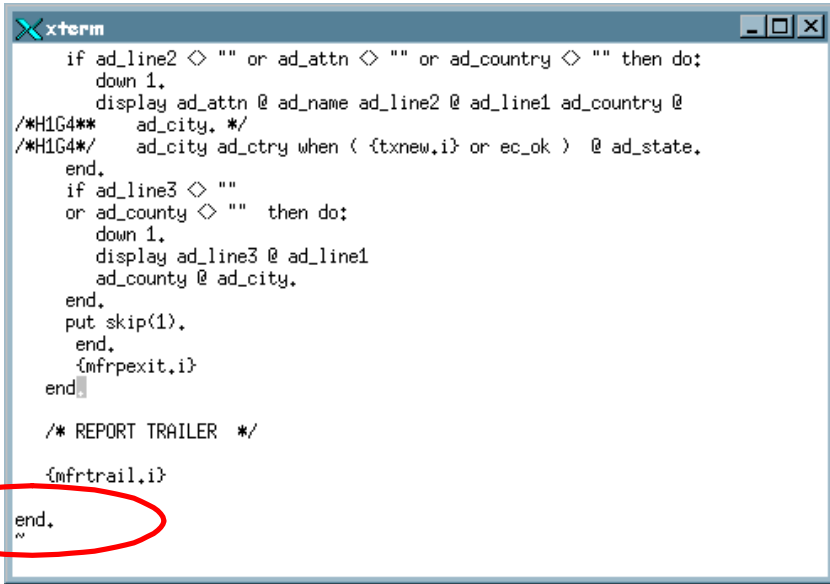
{mfreaset.i {1}}

{mfmsg.i 9 1}

mfrtrail.i and mfrl080.i


Our Passion. Your Advantage.

mfrtrail.i & mfrl080.i – to use...



```
xterm
if ad_line2 <> "" or ad_attn <> "" or ad_country <> "" then do:
  down 1.
  display ad_attn @ ad_name ad_line2 @ ad_line1 ad_country @
/*H1G4**  ad_city. */
/*H1G4*/  ad_city ad_ctry when ( {txnew,i} or ec_ok ) @ ad_state.
end.
if ad_line3 <> ""
or ad_county <> "" then do:
  down 1.
  display ad_line3 @ ad_line1
  ad_county @ ad_city.
end.
put skip(1).
end.
{mfrpexit,i}
end.

/* REPORT TRAILER */

{mfrtrail,i}

end.
~
```

QAD Proprietary

BPC-05-RE-530

Exercise 2



Exercise 2 (30 minutes)

- ▲ Rewrite the query from the previous exercise in using the supplied report template (test with output to terminal)
- ▲ Use the std. QAD Enterprise Applications includes (for a 132 report)
 - mfnpr.i (next-prev)
 - mfquoter.i (batch)
 - mfselbpr.i (output to)

The screenshot shows a QAD report template. The top section is a main menu with the following text:

```

progpr03 qrp3 .0 Main Menu 03/06/01
=====
From: 10-10000 To: 22-100
Output: printer
Batch ID:

```

A large black arrow points from the main menu to a detailed order line item section. This section contains the following text:

```

Item Number: 10-10000 Airco Industrieeel Model Prod Line: 3000

Order  Sold-To  Ln  Qty Ordered Qty to Invoice
-----
S9900001 cust1    1      5,0      0,0

Item Number: 10-15000 Airco Automotive Prod Line: 3000

```

QAD Proprietary

BPC-05-RE-544

Don't forget to use {fromedit.i} due to online translations.

Exercise 3



Exercise 3 (15 minutes)

- ▲ Create the following simple character program using the new message include pxmsg.i with message number 1 (adding new record)
 - {mfdeclre.i}
 - {pxmsg.i &MSGNUM=1}
- ▲ What seems to be the problem, solutions ???
- ▲ **Please wait for further instructions**
- ▲ If you still do not see anything, try the following:
 - {mfdeclre.i}
 - {pxmsg.i &MSGNUM=1
 - &PAUSEAFTER=TRUE}
- ▲ Experiment with the other parameters (check out the include pxmsg.I)

Field Help



Field Help

- ▲ Field help is accessed by pressing the Help key (F2 in ChUI, F1 in GUI) which executes a program called applhelp.p
 - applhelp.p accesses the flh_mstr database file to determine what scrolling window help program to execute
- ▲ Field help is added or changed in mgflhmt.p

flh_mstr	
flh_field	Dictionary Field or Code Variable
flh_call_pgm	Calling Procedure
flh_exec	Procedure to execute
flh_y	y coordinate
flh_x	x coordinate

Window Help

QAD
Our Passion. Your Advantage.

Window Help Maintenance

If the name of the calling procedure is left blank, field help is available for the field in all procedures

The gplu* procedures are created in the Browse Maintenance Menu. 80% of the scrolling windows from 72/73/74 were converted to lookup browses in 85.

267, 38
VT220-7 -- ohsu04 via TELNET
Num

QAD Proprietary
BPC-05-RE-57

Record Sequencing



Record Sequencing (1 of 2)

▲ mfnctrlc.i

- retrieves the next control file number with a prefix
 - {1} control file name
 - {2} control file prefix field
 - {3} control file number field
 - {4} table
 - {5} field
 - {6} variable/field to set
 - {7} condition expression

▲ sosomta1.p

```
/* GET NEXT SALES ORDER NUMBER WITH PREFIX */  
{mfnctrlc.i soc_ctrl soc_so_pre soc_so so_mstr  
so_nbr sonbr}
```



Record Sequencing (2 of 2)

▲ mfnctrl.i

- retrieves the next control file number

{1} control file name

{2} control file field

{3} file

{4} field

{5} variable/field to set

- ### ▲ wowomt.p (provided that the auto work order number flag is set to *Yes* in the work order control file)

```
/*GET NEXT WORK ORDER NUMBER*/
```

```
{mfnctrl.i woc_ctrl woc_nbr wo_mstr wo_nbr wonbr}
```

E-mail Definition Maintenance



Our Passion. Your Advantage.

E-mail Definition Maintenance [mgemmt.p]

E-Mail Definition Maintenance X

Go To Actions Copy Print Preview

E-mail System: unix

Operating System: unix

Start Effective: 11/19/2008

Description: internal email

Path and Program Name:

Command Line Parameters:

Sender:

Recipient:

Subject:

Message Text File:

Message Text String:

Start Effective: 11/19/2008

End Effective:

E-Mail Command:

Valid entries are UNIX, MSDOS and WIN32. This is the Progress name for the operating system of the computer where the user runs QAD Enterprise Applications

After you have entered the appropriate operating system, path and program name, parameters, and parameter sequence, E-Mail Definition Maintenance displays the command line at the bottom of the screen. At least one user must be linked to this record before a valid test message can be sent.

QAD Proprietary

E-Mail Definition Maintenance (36.4.20)

BPC-05-RE-600





E-mail and QAD Enterprise Applications (1 of 2)

- ▲ E-Mail Definition Maintenance (mgemmt.p)
builds a command line to serve as an interface
between QAD Enterprise Applications and your
e-mail system
- ▲ Examine the documentation on your e-mail
system to determine the requirements of the
command
- ▲ See if the e-mail system provides a command-
line interface
 - If it does not, shareware programs are available that
provide the required command-line interface



E-mail and QAD Enterprise Applications (2 of 2)

- ▲ E-Mail Definition Maintenance lets you define more than one e-mail system if required
 - You must enter an e-mail system code and e-mail address for each user in User Master Maintenance
- ▲ When you assign codes for more than one e-mail system, the codes do not have to be unique
 - If some of your users log onto QAD Enterprise Applications from different systems, for instance, you can set up an e-mail definition code of 200 for a UNIX system and another code 200 for a Windows system
 - Defining the multi-system user's e-mail system as 200 gives that user access to both e-mail systems, depending on where they log on

User Maintenance

QAD
Our Passion. Your Advantage.

User Maintenance [mgumt.p]

User Maintenance x

Go To Actions Copy Print Preview

User ID: mef User Name: Mark Farley

Language: Country Code: User Type: Employee Time Zone: PST/PDT E-mail Def: E-mail Address: mef@qad.com Variant: Restricted: Access Location: PRIMARY

Menu Style: A (A - Icons B - Tear Off C - Character) Menu Substitution: Remark:

System Access

Active: Active Reason: Force Password Change: Last Logon: Last Password Change:

Delete Back Next

Now you can send a test message from E-mail Definition Maintenance

Set up your userid and language and then go to the email folder and enter your email address (e.g.: fxd@qad.com) & enter the e-mail definition

User Maintenance (36.3.1)

QAD Proprietary BPC-05-RE-630

Event Notification



Event Notification

▲ Standard product

- Built into the PCC Module
 - Notify the next person on the approval routing
- Incorporated into the GRS Design
 - Notify Purchasing when the requisition has been approved

▲ Potential custom use

- Use as toolbar button
 - Send e-mail to next person in workflow
- Use with database triggers
 - Send e-mail to Purchasing when inventory falls below reorder point

QAD Proprietary

BPC-05-RE-644

PCC = Product Change Control

GRS = Global Requisition System

Review



Functionality Reuse

- ✓ Identify programs in QAD Enterprise Applications that can be re-used for customizing
 - Printing
 - Batch submittal
 - Screen/Report headers and trailers
 - Scrolling look-up windows
 - E-mail

Chapter 6

Database Schema

Introduction



Database Schema

In this section, you learn how to:

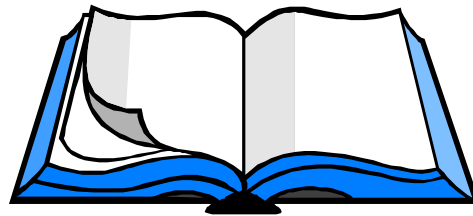
- ▲ Identify how the database is set up
- ▲ Identify features of the database
- ▲ Freeze and unfreeze databases
- ▲ Add new schema objects
- ▲ Create a database

Terminology



Terminology

- ▲ DDL
- ▲ CRC



QAD Proprietary

BPC-06-SCH-020

Data Definition Language (DDL) . This is a language used to describe changes to the schema to the Progress DBMS. Progress provides tools for reading and writing these command files. All schema changes are in this form.

Cyclical Redundancy Check (CRC). An algorithm used by PROGRESS to verify the integrity of a source file by comparing an internally tagged number with the same number in a called file or database schema. PROGRESS generated r-code contains the CRC value of the schema against which it was compiled. Whenever the schema is modified and a new CRC value is produced, those programs containing the old CRC value must be re-compiled or PROGRESS will consider the programs invalid when executed.

Standard Databases

 QAD Enterprise Applications Standard Databases	
Database	Notes
mfgempty & mfg	The mfgempty database consists of only the QAD Enterprise Applications schema with no system data. The mfg database is a copy of mfgempty with the data from the mfg directory loaded. The logical name must be qadddb.
hlpempty & mfghelp	A single help database is usually shared by a number of production or test databases. The qadhelp database is a copy of hlpempty with the help data loaded. The logical name must be qadhelp.
admempty & mfgadm	The admin database includes all the data related to the User Interface and other information not directly related to the business logic. The admin database is a copy of admempty with the data from the admin directory loaded. The logical name must be qadadm.
mfgtrain	A copy of mfgempty with the data from the mfgtrain directory loaded.
mfgdemo	A copy of mfgempty with the data from the mfgdemo directory loaded.

The qadadm database contains information about the browses, UI, EDI setups, Desktop menu program links, labels and translations and any other tables that are not related to business logic (qadddb db) or help (qadhelp db).



QAD Enterprise Applications Schema Documentation

- ▲ Data Definitions
- ▲ File Relationships

QAD Enterprise Applications Data Definitions



Data Definition Notations

- ◆ New file, field, index
- ▲ Changed field/index
- * Primary index
- ! Column-label (second line initiation)
- × Obsolete field (<prefix> _X<name>)
- [x] Extent of array field

Database Definitions Manual

File/Field Definitions

so_mstr Sales Order Master (partial listing)

Field Name	Label	Col-label	Format	Dec
so_user1	Ufld1		x(8)	
so_user2	Ufld2		x(8)	
so_userid	User ID		x(8)	
so_weight	Weight		->>>>>>>>>9.9999	4
so_weight_um	Weight UM	UM	x(2)	
so_xcomm_pct[2]	Commission	Comm%	>>9.99%	2
so_slspsn[2]	Salesperson	Salespsn	x(8)	
so_chr01			x(8)	
so_chr02			x(8)	
so_chr03			x(8)	
so_chr04			x(8)	
so_chr05			x(8)	
so_chr06			x(8)	
so_chr07			x(8)	
so_chr08			x(8)	
so_chr09			x(8)	
so_chr10			x(8)	
so_dec01			>>>>>>>>>9.99	2
so_dec02			>>>>>>>>>9.99	2
so_dte01			99/99/99	
so_dte02			99/99/99	
so_log01			yes/no	
so_qad01			-z,zzz,zzz,zz9.99	2
so_qad02			-z,zzz,zzz,zz9.99	2
so_qad03			-z,zzz,zzz,zz9.99	2
so_qad04			>	
so_qad05[2]			>>9.9<<	2

Database Definitions Manual

Index Definitions

so_mstr

Sales Order Master

Index Name	Unique	Field-Name	Ascending	Abbreviate
so_cust	no	so_cust	yes	no
so_due_date	no	so_due_date	yes	no
so_invoice	no	so_inv_nbr	yes	no
*so_nbr	yes	so_nbr	yes	no
so_po_nbr	no	so_po	yes	no
		so_cust	yes	no
		so_nbr	yes	no
so_quote	no	so_quote	yes	no
		so_nbr	yes	no

Index Definitions

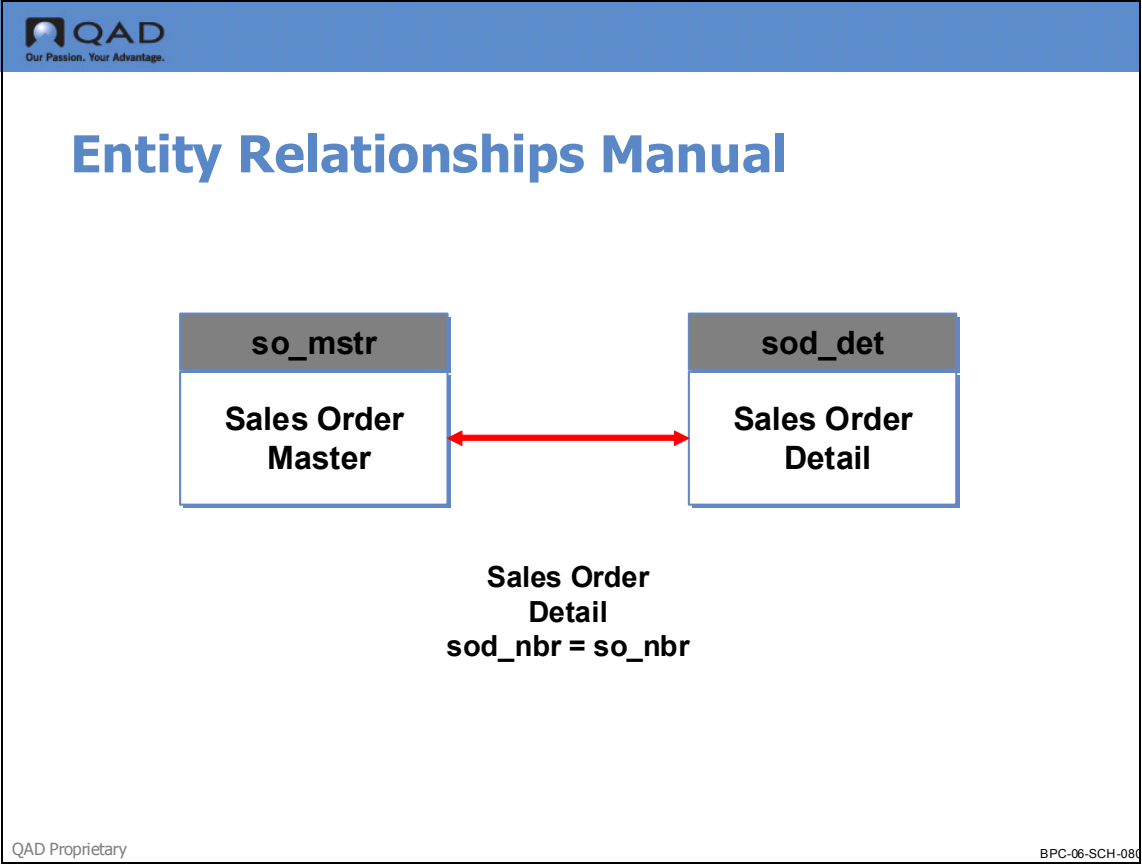
so_det

Sales Order Detail

Index Name	Unique	Field-Name	Ascending	Abbreviate
*sod_nbrln	yes	sod_nbr	yes	no
		sod_line	yes	no
sod_part	yes	sod_part	yes	no
		sod_nbr	yes	no
		sod_line	yes	no

KEY: * primary index

QAD Enterprise Applications File Relationships





QAD Enterprise Applications Database Features

- ▲ User-definable database fields in existing files
- ▲ User Workfile

User-Definable Fields

Our Passion. Your Advantage. Existing User-Definable Fields					
	Field Name	Label	Col-label	Format	Dec
User Definable	so_user1	Ufld1		x(8)	
	so_user2	Ufld2		x(8)	
	so_userid	User ID		x(8)	
User Definable	so_chr01			x(8)	
	so_chr02			x(8)	
	so_chr03			x(8)	
	so_chr04			x(8)	
	so_chr05			x(8)	
	so_chr06			x(8)	
	so_chr07			x(8)	
	so_chr08			x(8)	
	so_chr09			x(8)	
	so_chr10			x(8)	
	so_dec01			>>>, >>>, >>9.99	2
	so_dec02			>>>, >>>, >>9.99	2
	so_dte01			99/99/99	
	so_dte02			99/99/99	
	so_log01			yes/no	
Reserved for QAD	so_qad01			-z,zzz,zzz,zz9.99	2
	so_qad02			-z,zzz,zzz,zz9.99	2
	so_qad03			-z,zzz,zzz,zz9.99	2
	so_qad04			>	
	so_qad05[2]			>>9.9<<	2

QAD Proprietary

BRU-10-SCH-100

User Workfile

QAD
Our Passion. Your Advantage.

Existing User Workfile

<u>Field Name</u>	<u>Format</u>	
usrw_charfld[15]	x(8)	usrw_wkfl Six key fields Three compounded indices, each with two key fields Arrays of following formats - Character - Date - Decimal

usrw_datefld[4]	99/99/99
usrw_decfld[15]	->>, >>9.99
usrw_key1	x(8)
usrw_key2	x(8)
usrw_key3	x(8)
usrw_key4	x(8)
usrw_key5	x(8)
usrw_key6	x(8)
usrw_user1	x(8)
usrw_user2	x(8)

<u>Index Name</u>	<u>Unique</u>	<u>Field-Name</u>
*usrw_index1	yes	usrw_key1 usrw_key2
usrw_index2	no	usrw_key3 usrw_key4
usrw_index3	no	usrw_key5 usrw_key6

QAD Proprietary
BPC-06-SCH-116

usrw_wkfl Example

```
create usrw_wkfl.  
usrw_key1 = "sosomt".  
usrw_key2 = sod_nbr + string(sod_line, "999").  
usrw_charfld[1] = transportation info.  
usrw_charfld[2] = transportation info.  
usrw_key3 = "sod_det".  
usrw_key4 = alternate_index_data.
```

Changing Database Schema



Changing the QAD Enterprise Applications Database

- ▲ Not encouraged
 - Hard to test standard QAD Enterprise Applications
 - Hard to maintain upgrades/patches
- ▲ If you must
 - Start with a copy
 - Always keep the original database intact
 - Freezing and unfreezing database files
 - Adding new schema objects
 - Indexes
 - Fields
 - Files (not quite yet)

Be Thorough

- ▲ Keep original mfgempty.db
- ▲ Make a new custom “empty” database
 - This will contain all schema modifications
- ▲ Always make a backup
 - Use the prodb command
 - Syntax

```
prodb [new-db-name]  
      [existing-db-name]
```

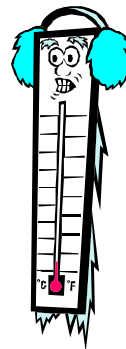
- ▲ Apply changes to all copies of database
 - Development database
 - Test database
 - Compile database
 - Production database
 - Apply changes in Single User Mode!

“Frozen” Databases



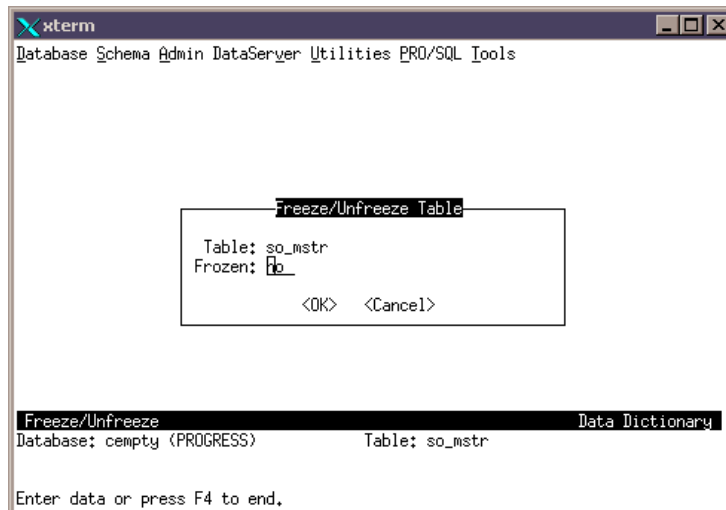
“Frozen” Databases

- ▲ Protects databases from schema modifications
- ▲ QAD Enterprise Applications databases are “frozen” by default
- ▲ Must “unfreeze” database files to
 - Add indexes (through v8)
 - Add fields
 - Modify fields
- ▲ Re-freeze database files when modifications are complete



Freeze / Unfreeze

- ▲ Data Dictionary Utilities Menu
 - One table at a time



Great if you are only modifying a few tables. Tedious if you are modifying many tables.

Freeze / Unfreeze

- ▲ Run program in the PROGRESS editor
 - One table at a time
 - Multiple tables at a time

```
FOR EACH _File EXCLUSIVE-LOCK
where _Frozen = YES and _Hidden = NO
/* and _File-Name = "so_mstr" */ :
    _Frozen = NO.
END.
```

Adding New Database Schema



Adding New Schema

- ▲ If you must
 - Keep track of all changes
 - Create a delta .df file



PROGRESS File/Field/Index Naming Issues

- ▲ Length can be 1 to 32 characters
- ▲ First character must be alphabetic (a-z or A-Z)
- ▲ Can be any combination of letters, numbers and special characters
 - special characters are: # \$ % & - _
 - PROGRESS/SQL does not allow the dash (-)

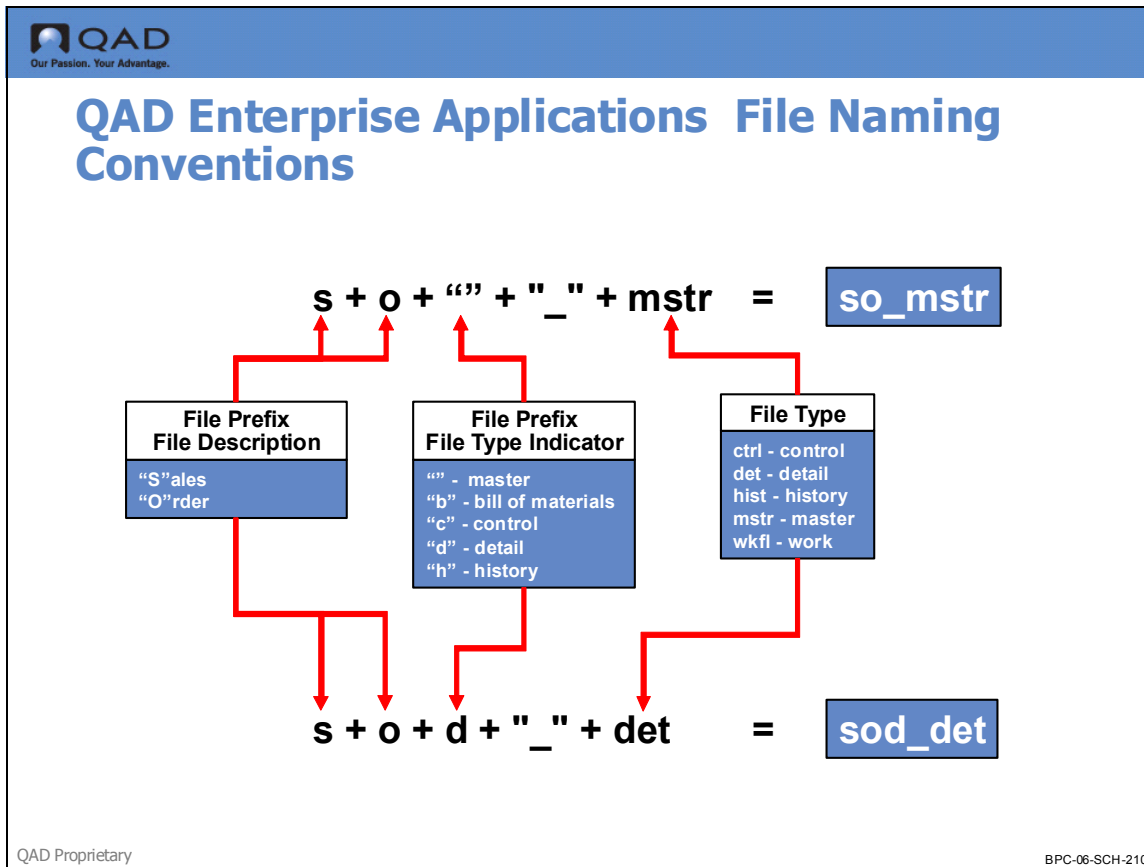
Schema Naming Conventions



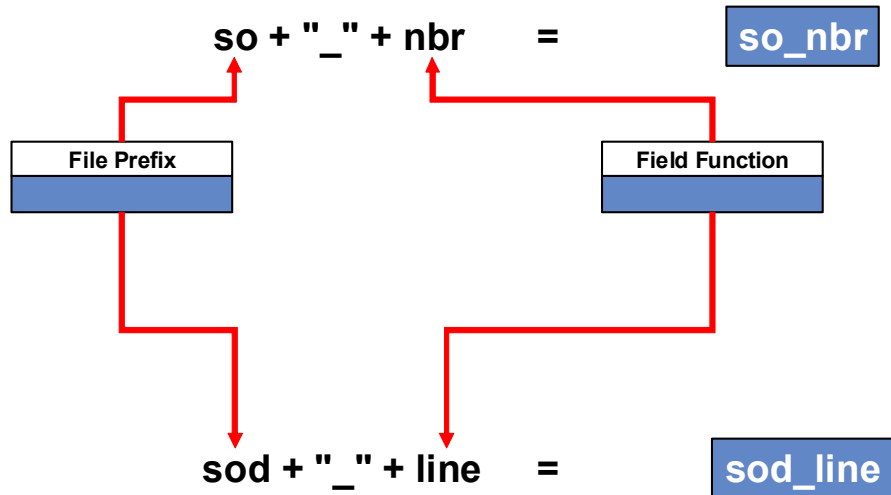
QAD Enterprise Applications Schema Naming Conventions

- ▲ Names consist of a prefix and a suffix separated by an underscore (_) character
- ▲ File Names
 - prefix = the first few characters of the file description
 - suffix = the file type (master, detail, workfile, control...)
- ▲ Field Names
 - prefix = the file name prefix
 - suffix = the function of the field
- ▲ Index Names
 - prefix = the file name prefix
 - suffix = the index sort

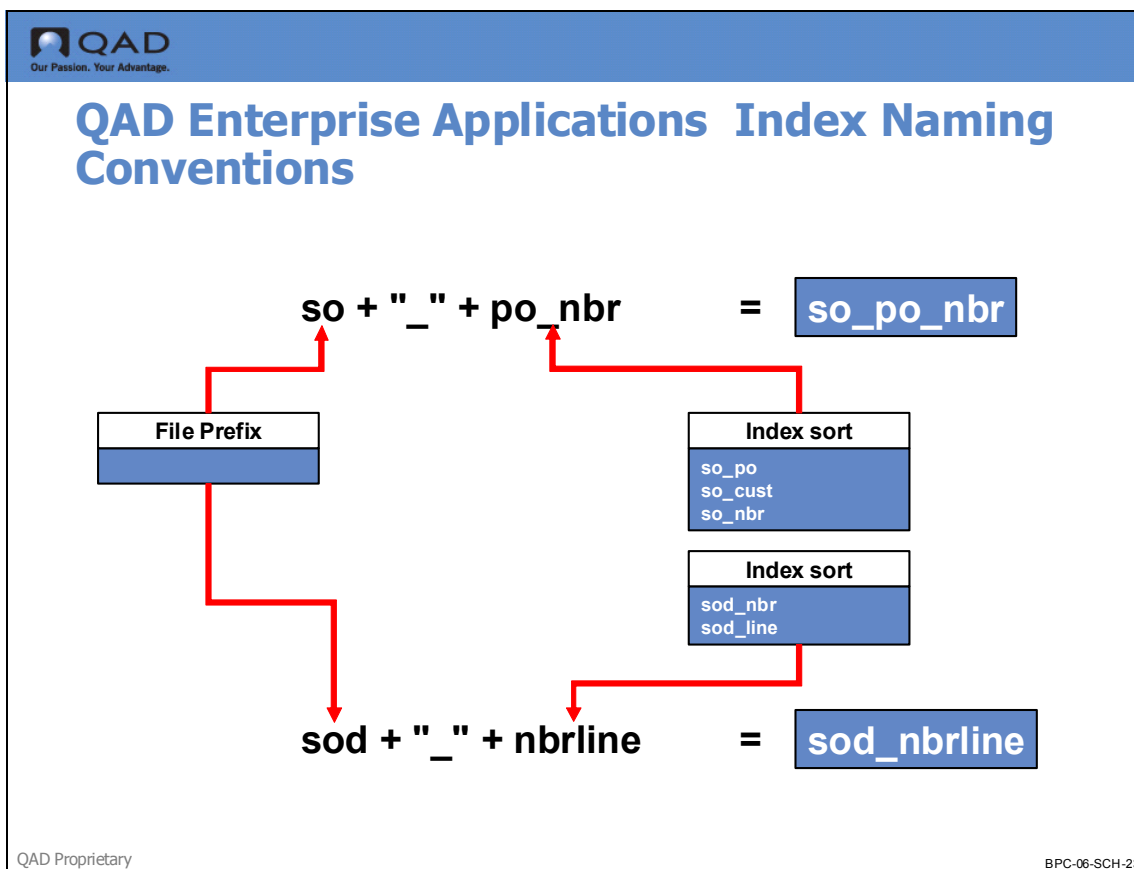
File Naming Conventions



QAD Enterprise Applications Field Naming Conventions



Index Naming Conventions



Adding Indexes



Adding Indexes

▲ Advantages

- Speeds up data access
- Avoids or limits record scanning operations
- Avoids sorting operations
- Supports common queries

▲ Disadvantages

- Consumes disk space
- Adds processing overhead of many data operations - add, update, delete, copy, and move
- Every update of an indexed field causes an update of the index and related indexes as well



Index Limits

▲ Index Entries

- Maximum of 16 fields per index
- Total variable-length storage requirements of all fields in an index entry must be less than 1000 bytes

▲ Maximum number of indexes

- In PROGRESS version 8, it depends on the database block size
 - block size of 1kB = 6400 indexes
 - 8kB = 32000 indexes
- Version 9 allows indexes up to 32K

Adding / Modifying Indexes

- ▲ Data Dictionary Schema Menu
 - Index editor

so_bill

Non-Unique ACTIVE

Seq	Field Name	Type	Asc	Abbr
1	so_bill	char	Asc	No
2	so_nbr	char	Asc	No
3	so_ord_date	date	Asc	No

Next Prev First Last Rename Add Delete ChangePrimary Uniqueness
MakeInactive Browse SwitchTable GoField ROWID Undo Exit

Database: empty (PROGRESS) Table: so_mstr

Look at the next index of this table.

Lists the indexes
for a selected
table

Lists the
components of
the index
selected on the
left



Adding / Modifying Indexes

▲ Creating new index "so_site"

xterm

Database Schema Admin DataServer Utilities PRO/SQL Tools

Adding Index "so_site"

Num	Field Name	Type	Asc?
1	so_site	char	yes

	so_shipvia	char	
	so_ship_date	date	
	so_ship_eng	logi	
	so_size	deci	
	so_size_um	char	
	so_source	char	
	so_stat	char	
	so_taxable	logi	
	so_taxc	char	
	so_tax_date	date	
	so_tax_env	char	

Database: empty (PROGRESS) Table: so_ms

Use [RETURN] to add component, [F1] to save, [F4] to undo.

Use the cursor-motion keys to navigate, or type field name and hilite will move.

As you select fields to be key components, they are moved above the dividing line. Use [RETURN] to select fields. To set asc/desc, use +/- keys.

To unselect fields, move hilite above dividing line and [RETURN]. When done, press [F1].

After providing a name for a new index (the new index name is **so_site**) select the fields that make up the index sort (the field **so_site** has been selected)

Adding / Modifying Indexes

- ▲ Creating new index "so_site" using DDL (Data Definition Language)

```
ADD INDEX "so_site" ON "so_mstr"  
INDEX-FIELD "so_site"  
ASCENDING ABBREVIATED
```

Adding Fields



Adding Fields

▲ Disadvantages

- Managing QAD Enterprise Applications version updates
 - Maintaining custom delta .df
 - Maintaining QAD Enterprise Applications delta .df
- Managing r-code

▲ Advantages

- New fields available within the existing table
 - No extra database accesses
- #### ▲ Better to add information to a “side” database
- Add a new table as an addendum to an existing table
 - Add new fields to the new “addendum” table

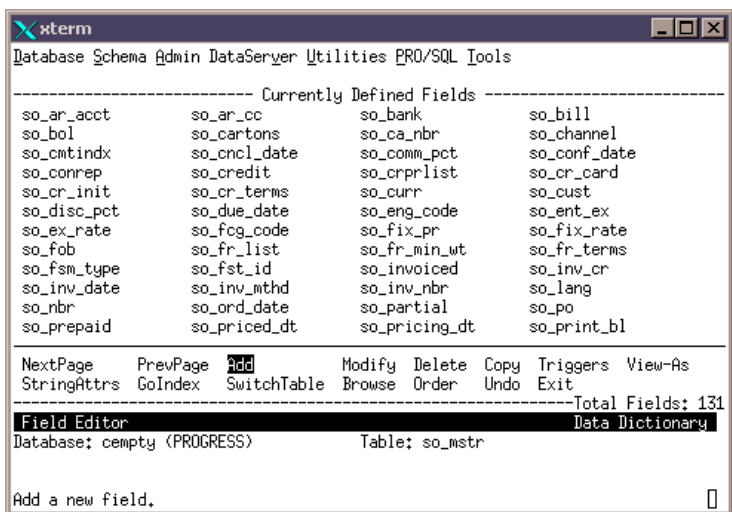
Field Limits

- ▲ PROGRESS allows 32,000 fields per database
- ▲ Maximum number of fields per table is 500
 - Careful - fields can add up fast!



Adding Fields (1 of 4)

- ▲ Data Dictionary Schema Menu
- Field Editor



The field editor lists the existing fields for the selected table

Adding Fields (2 of 4)

- ▲ Data Dictionary Schema Menu
 - Field Editor

Field-Name: so_custom

Format: x(8)

Label: Custom

Column-label: ?

Initial:

Component of-> View: no

Valmsg:

Help:

Desc: Custom Field

Data-Type: character

Extent:

Decimals: ?

Order: 1590

Mandatory: no (Not Null)

Case-sensitive: no

NextPage

PrevPage

Add

Modify

Delete

Copy

Triggers

View-As

StringAttrs

GoIndex

SwitchTable

Browse

Order

Undo

Exit

Field Editor

Database: empty (PROGRESS)

Table: so_mstr

Total Fields: 131

Data Dictionary

Enter data or press F4 to end.

To add a field you must provide a name and a datatype.

Formats default depending on the datatype but can be modified.



Adding Fields (3 of 4)

- ▲ Creating new field "so_custom" using DDL

```
ADD FIELD "so_custom" OF "so_mstr"  
  AS character  
  DESCRIPTION "Custom Field"  
  FORMAT "x(15)"  
  INITIAL ""  
  LABEL "Custom"  
#    ORDER 1590
```

Adding Files



Adding Fields (4 of 4)

- ▲ New files to support customizations
- ▲ New files to augment existing files
- ▲ CREATE IN A SIDE DATABASE
 - Keep custom files separate from QAD Enterprise Applications files

Side Databases



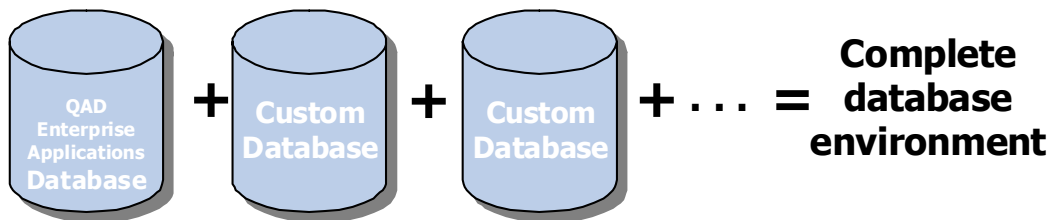
QAD Enterprise Applications Side Databases

- ▲ Rationale for side databases
- ▲ Creating a side database
- ▲ Addendum files

Rationale for Side Databases

- ▲ Reduce CRC conflicts
- ▲ Isolate customizations
- ▲ Eliminate data conversion problems when upgrading
- ▲ Retain standard set of dump/load procedures

Database Environment



Database Naming Conventions



PROGRESS Database Naming Issues

- ▲ First character must be alphabetic (a-z or A-Z)
- ▲ Cannot contain PROGRESS reserved words
- ▲ Cannot use special characters:
 \ " ` * ; | ? [] () - ! \$ { } < >
- ▲ Cannot be QADDB or QAD
- ▲ Maximum length
 - UNIX using HPFS 1 to 11 characters
 - Windows using NTFS 1 to 11 characters
 - Windows using FAT 1 to 8 characters
- ▲ Pathname length limits
 - UNIX 1 to 60 characters
 - Windows using FAT 1 to 63 characters

Creating Databases



Creating Databases

- ▲ Data Dictionary
- ▲ PRODB



Creating Databases

- ▲ Data Dictionary Database Menu
 - Create



Provide a new database name and choose what database you to copy.

New databases are always a copy of some existing database (most often they are a copy of the PROGRESS empty database.



Creating Databases

▲ prodb Command

– Syntax:

```
prodb [new-db-name]  
      [empty | sports | existing-db-name]
```

```
/dr02/mfgpro/db>$DLC/bin/prodb custldb /dr01/progress/dlc91d/empty  
New PROGRESS database custldb created (from empty).
```

QAD Proprietary

BPC-06-SCH-410

- Prodb creates a single volume database
- Procopy copies an existing database

Note Do not use an operating system utility to copy a Progress database.

Adding Database Files



Adding Database Files

- ▲ Create addendum files in a side database to augment the fields to standard QAD Enterprise Applications database files
- ▲ Create new files in a side database to support customization needs



Addendum Files

File:

standard file name
so_mstr

addendum file name
c1so_mstr

Keeping the names similar to the tables they are augmenting will allow you to easily associate the original and the custom tables.

Field:

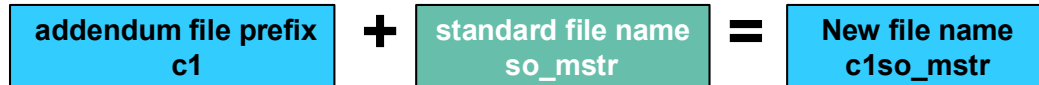
standard field name
so_nbr

addendum field name
c1so_nbr

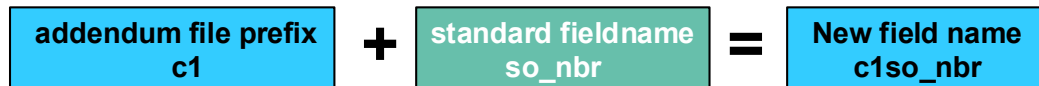
Using the same names to identify the unique key fields allows easy association to the custom table.

Naming Addendum Files

File: c + n + standard file name



Field: Addendum file prefix + standard field name



Naming New Custom Files

File: c + n + X + "_" + ZZZZ = c2_mstr

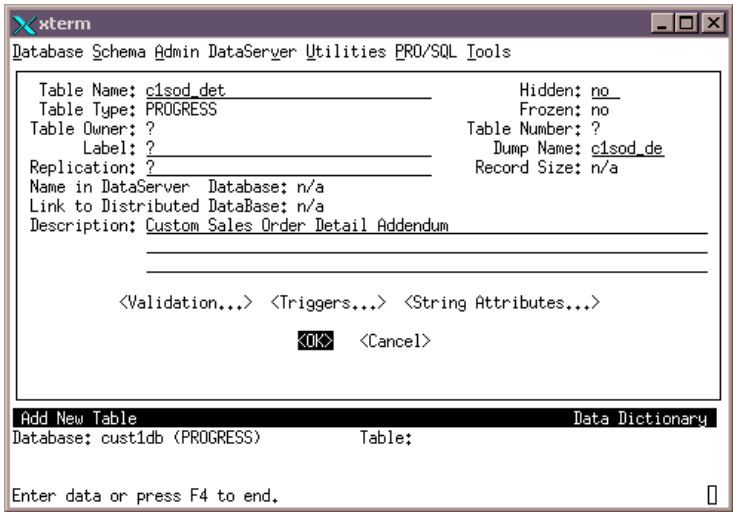
File Prefix File Type Indicator
"m" - master
"b" - bill of materials
"c" - control
"d" - detail
"h" - history

File Type
ctrl - control
det - detail
hist - history
mstr - master
wkfl - work

Field: file prefix + "_" + function = c2_nbr

Adding New Files

- ▲ Data Dictionary Schema Menu
 - Add New Table



xterm

Database Schema Admin DataServer Utilities PRO/SQL Tools

Table Name: cisod_det Hidden: no
Table Type: PROGRESS Frozen: no
Table Owner: ? Table Number: ?
Label: ? Dump Name: cisod_de
Replication: ? Record Size: n/a
Name in DataServer Database: n/a
Link to Distributed DataBase: n/a
Description: Custom Sales Order Detail Addendum

<Validation...> <Triggers...> <String Attributes...>
<OK> <Cancel>

Add New Table Database: cust1db (PROGRESS) Table: Data Dictionary

Enter data or press F4 to end.

Be sure to describe the table's function in the description field

After choosing OK, the field editor Add function is automatically run

Adding New Files

▲ Creating new file "c1sod_det" using DDL

```
ADD TABLE "c1sod_det"  
  LABEL "SO Detail Addendum"  
  DUMP-NAME "c1sod_de"  
  
ADD FIELD "c1sod_nbr" OF "c1sod_det" AS character  
  FORMAT "x(8)"  
  INITIAL ""  
  LABEL "SO Number"  
  ORDER 10  
  
ADD FIELD "c1sod_line" OF "c1sod_det" AS integer  
  FORMAT ">>>>9"  
  INITIAL "0"  
  LABEL "Line Number"  
  COLUMN-LABEL "Line"  
  ORDER 20
```


Database Definition Files



Database Definition Files

- ▲ Used to re-create database schema from a PROGRESS empty database
 - Port databases to different OS Platforms
 - Create “empty” application databases

Creating Data Definition File



Creating a .df File

- ▲ Data Dictionary Admin Menu
 - Dump Data and Definitions
 - Data Definitions (.df file)

Database Schema Admin DataServer Utilities PRO/SQL Tools

Table Name: (or "ALL" for all tables)
ALL

c1sod_det
c1so_mstr
c1_mstr

Data Definitions (.df file) Database: cust1db (PROGRESS) Table: c1so_mstr Data Dictionary

Press the [F4] key to end.

Can dump the definitions of individual tables or all tables

Dumping Data Definitions



Dump Database Definitions

▲ Naming the data definitions (.df) file

xterm

Database Schema Admin DataServer Utilities PRO/SQL Tools

Dump Data Definitions for All Tables

Output File: cust1db.df <Files...>

Code Page: iso8859-1

<OK> <Cancel>

Data Definitions (.df file)	
Database: cust1db (PROGRESS)	Table: Data Dictionary

Enter data or press F4 to end.

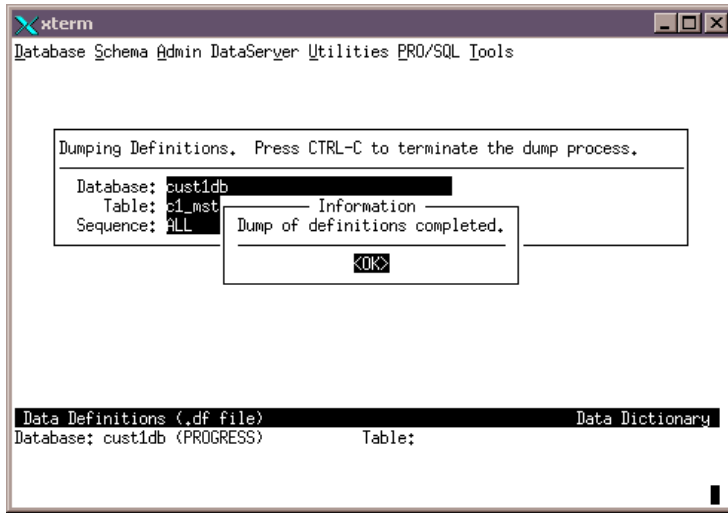
Supply the name of the definition file. Defaults to the database name

Don't change the code page!

 **QAD**
Our Passion. Your Advantage.

Dumping Definitions

▲ Data Definitions (.df file)



xterm

Database Schema Admin DataServer Utilities PRO/SQL Tools

Dumping Definitions. Press CTRL-C to terminate the dump process.

Database: cust1db
Table: cl_mst
Sequence: ALL

Information
Dump of definitions completed.
OK

Data Definitions (.df file) Data Dictionary
Database: cust1db (PROGRESS) Table:

**Screen
allows you
to monitor
the progress**

QAD Proprietary

BPC-06-SCH-510

Database Definition (.df) File

```
UPDATE DATABASE "?"                                     Cust1db.df

ADD TABLE "clso_mstr"
  DESCRIPTION "Custom Sales Order Master Addendum"
  DUMP-NAME "clso_mst"

ADD FIELD "clso_nbr" OF "clso_mstr" AS character
  FORMAT "x(8)"
  INITIAL ""
  LABEL "SO Number"
  COLUMN-LABEL "SO Num"
  ORDER 10

ADD FIELD "clso_custom" OF "clso_mstr" AS character
  FORMAT "x(8)"
  INITIAL ""
  LABEL "Custom"
  ORDER 20

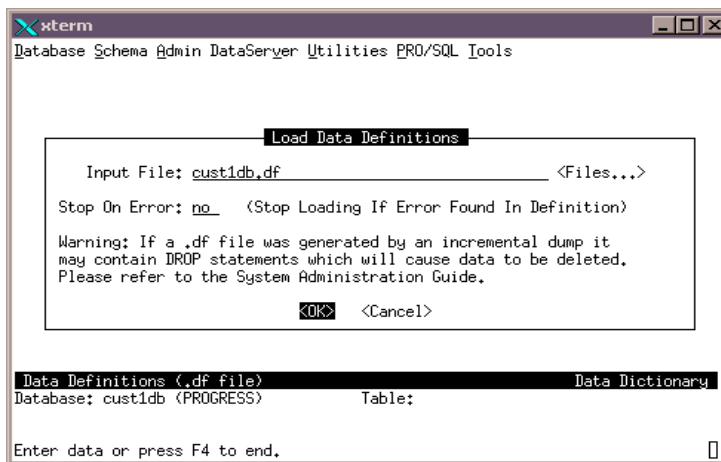
ADD INDEX "clso_nbr" ON "clso_mstr"
```

Loading Database Definitions



Loading Database Definitions

- ▲ Data Dictionary Admin Menu
 - Load Data and Definitions
 - Data Definitions (.df file)

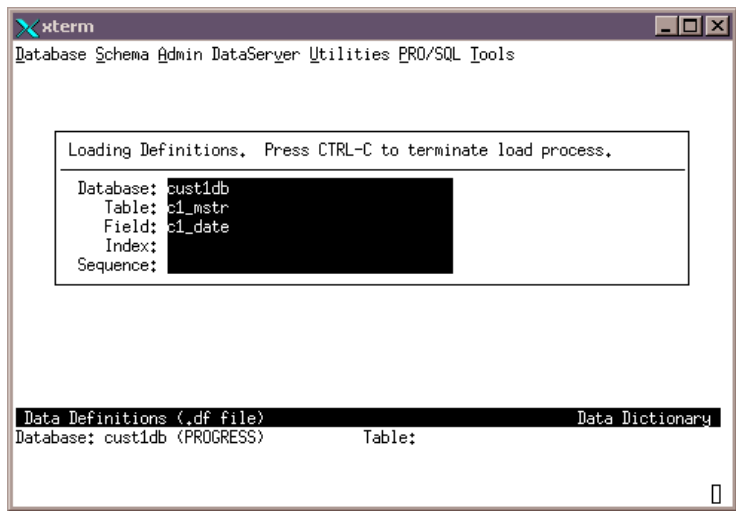


Supply the name of the definition file. Defaults to the database name

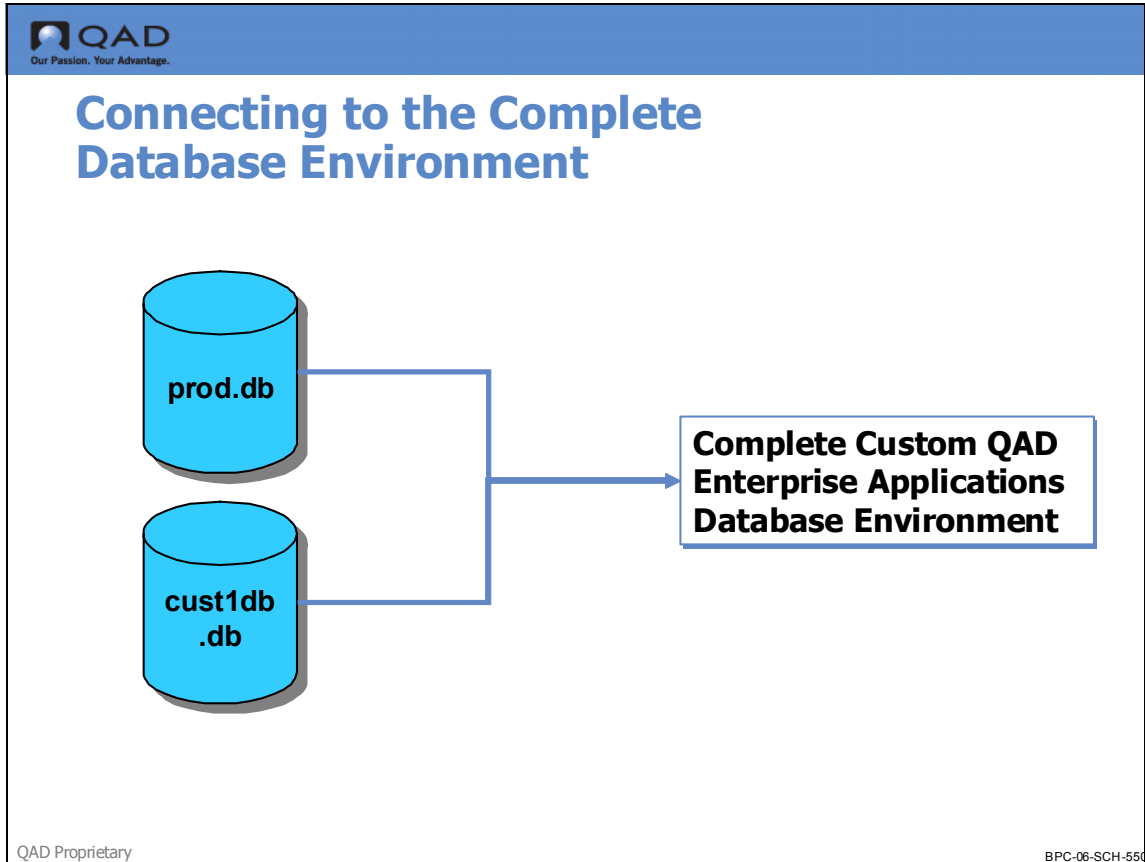
NOTE: Load definitions into an empty database

Loading Definitions

▲ Data Definitions (.df file)



**Screen
allows you
to monitor
the progress**



Database Connection Options



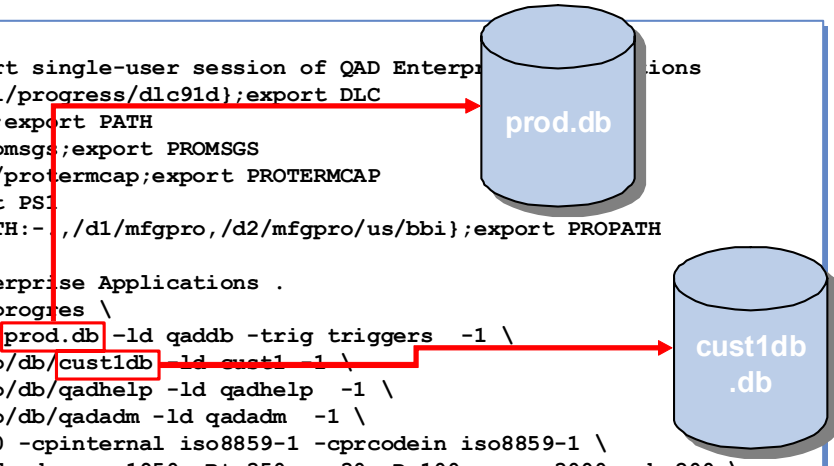
Database Connection Options

- ▲ Startup script
- ▲ Programmatic

Startup Script DB Connection

```
#!/bin/sh
# Script to start single-user session of QAD Enterprise Applications
DLC=${DLC:-/dr01/progress/dlc91d};export DLC
PATH=$PATH:$DLC;export PATH
PROMSGS=$DLC/promsgs;export PROMSGS
PROTERMCAAP=$DLC/protermcap;export PROTERMCAAP
PS1='$$ ';export PS1
PROPATH=${PROPATH:-, /d1/mfgpro, /d2/mfgpro/us/bbi};export PROPATH

# Start QAD Enterprise Applications .
exec $DLC/bin/progress \
/dr02/mfgpro/db/prod.db -ld qaddb -trig triggers -1 \
-db /dr02/mfgpro/db/cust1db -ld cust1 -1 \
-db /dr02/mfgpro/db/qadhelp -ld qadhelp -1 \
-db /dr02/mfgpro/db/qadadm -ld qadadm -1 \
-cpstream ibm850 -cpinternal iso8859-1 -cprcodein iso8859-1 \
-cpcoll basic -d mdy -yy 1950 -Bt 350 -c 30 -D 100 -mmax 3000 -nb 200 \
-s 63 -p mf.p;
```



Programmatic DB Connection

```
/* connect.p */  
connect prod -ld qadddb -1  
  -db cust1db -ld cust1 -1  
  -db qadhelp -ld qadhelp -1  
  -db qadadm -ld qadadm -1.
```

Can use a single
CONNECT
statement or
multiple **CONNECT**
statements

```
if connected(qadddb) and  
  connected(cust1) and  
  connected(qadhelp) and  
  connected(qadadm)  
then  
  run mf.p.
```

CRC Issues



CRC Issues

- ▲ At runtime, PROGRESS must ensure that r-code files and databases that they access are compatible.
 - If the schema has changed since the code was compiled, running the code might damage the database
- ▲ CRC validation
 - For each table, a CRC value is computed from selected elements of the table metaschema. At compile time, PROGRESS inserts the database CRC into the r-code for each table accessed by the procedure.
 - CRC values change only when certain schema elements critical to field or record definition change
 - As long as the table is structurally compatible with the r-code that references it, the CRC values match and the r-code can execute

CRC Calculation

- ▲ Metaschema fields in CRC Calculations
 - A change to any of the listed fields automatically changes the value of the CRC

_File	_Field	_Index	_Index-Field
_File-Name	_Field-Name	_Index-Name	_Field-Name
_DB-Lang	_Data-Type	_Unique	_Ascending
	_Dtype	_Num-Comp	_Abbreviate
	_Sys-Field		_Unsorted
	_Field-Rpos		
	_Decimals		
	_Order		
	_Extent		
	_Fld-Stdtype		
	_Fld-Stlen		
	_Fld-Stoff		
	_Fld-Case		

Field Validation

▲ Schema defined

The screenshot shows the 'Field Editor' window in QAD. The field being edited is 'c1sod_line' with data type 'integer'. The 'Valexp' (validation expression) is set to 'c1sod_line > 0', which is highlighted with a red box. A red arrow points from this box to a callout box on the right. The 'Valmsg' (validation message) is 'ERROR: sod_line must be greater than zero'. The window also shows a menu bar with options like 'NextPage', 'PrevPage', 'Add', 'Modify', 'Delete', 'Copy', 'Triggers', 'View-As', 'StringAttrs', 'GoIndex', 'SwitchTable', 'Browse', 'Order', 'Undo', and 'Exit'. At the bottom, it indicates 'Database: cust1db (PROGRESS)' and 'Table: c1sod_det'.

- Must evaluate to a logical value
- Can be a validation file (.v) or an expression

Schema Pre-Selection Dialog

- ▲ Object Name – Enter a PROGRESS wildcard (*) to filter object names
- ▲ Object Type – Leave it as * or specify one object type to filter (table, view, sequence, procedure, synonym)
- ▲ Object Owner – Enter a PROGRESS wildcard to filter owner names
- ▲ PROGRESS wildcard – * matches anything, ? matches one character



What to Do With the Output

- ▲ Keep the .SQL
- ▲ Dump the database definitions of the schema holder (.df)
- ▲ Shutdown the “throw-away” instance and delete the database files
- ▲ Alter the .SQL
 - to specify tablespaces and storage clauses
 - to fix any variable length problems
 - to remove NOT NULL specifications
 - to remove MANDATORY clauses

Exercise: Adding Schema:



Lab: Adding Schema

- ▲ Create an addendum table to the table that stores the customer contact information. Use this new table to store additional contact information like an e-mail address.
 - Create a new custom side database
 - Add a new table, new fields, and a new index
 - What are the benefits to adding custom schema to a side database?

Adding a Side Database



Solutions: Adding a Side Database

- ▲ Adding information to a side database
 - Things to think about:
 - What will we call the side database?
 - Suggestion, cust1.db
 - What are the key fields in the AD_MSTR table that we need to use for the addendum AD_MSTR table?
 - AD_MSTR.AD_ADDR
 - What will we call the addendum AD_MSTR table?
The key field? The new data fields?
 - Follow the standard, c1ad_mstr
 - New_prefix + original_name = C1 + AD_ADDR = C1AD_ADDR
 - C1AD_EMAIL and C1AD_EMAIL2

Solutions: Adding a Side Database

▲ Adding information to a side database

- Create the database

```
prodb cust1 empty
```

- Connect to cust1 single-user

```
$DLC/bin/_progres cust1 -1
```

- Run the Data Dictionary



Solutions: Adding a Side Database

- ▲ Adding information to a side database
 - Select Schema, Add New Table

Table Name: clad_mstr
Table Type: PROGRESS
Table Owner: ?
Label: ?
Replication: ?
Name in DataServer Database: n/a
Link to Distributed DataBase: n/a
Description: Custom Address Master Addendum

Hidden: no
Frozen: no
Table Number: ?
Dump Name: clad_mst
Record Size: n/a

<Validation...> <Triggers...> <String Attributes...>
<OK> <Cancel>

Add New Table
Database: cust1 (PROGRESS) Table:
Data Dictionary

Enter data or press F4 to end.

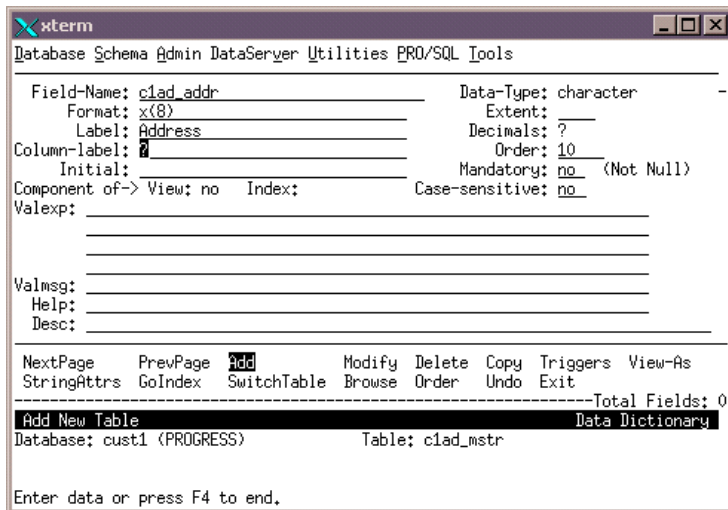
Enter table name

Accept defaults
(Hidden, Frozen,
Dump Name)

Enter a description

Solutions: Adding a Side Database

- ▲ Adding information to a side database
 - Automatically runs Field Editor, Add fields



The screenshot shows a terminal window titled 'xterm' with a menu bar: Database Schema Admin DataServer Utilities PRO/SQL Tools. The main area displays the 'Field Editor' for a field named 'c1ad_addr'. The configuration is as follows:

Field-Name: c1ad_addr	Data-Type: character
Format: x(8)	Extent: _____
Label: Address	Decimals: ?
Column-label: 2	Order: 10
Initial: _____	Mandatory: no (Not Null)
Component of-> View: no	Index: _____
Case-sensitive: no	
Valexp: _____	
Valmsg: _____	
Help: _____	
Desc: _____	

Below the form is a menu bar with the following options: NextPage, PrevPage, Add, Modify, Delete, Copy, Triggers, View-As, StringAttrs, GoIndex, SwitchTable, Browse, Order, Undo, Exit. The 'Add' option is highlighted. At the bottom, it says 'Total Fields: 0' and 'Add New Table'. Below that, it shows 'Database: cust1 (PROGRESS)' and 'Table: c1ad_mstr'. At the very bottom, it says 'Enter data or press F4 to end.'

**Add C1AD_ADDR
(defined like
AD_ADDR in
AD_MSTR)**

**When the
information for a
field is complete,
press the GO key
(F1 in ChUI/F2 in
GUI)**



Solutions: Adding a Side Database

- ▲ Add more fields by pressing the GO-Key
 - Press the END-KEY when all the fields have been added

xterm Database Schema Admin DataServer Utilities PRO/SQL Tools

Field-Name: clad_email Data-Type: character
 Format: x(45) Extent: _____
 Label: E-Mail Decimals: ?
 Column-label: ? Order: 20
 Initial: _____ Mandatory: no (Not Null)
 Component of-> View: no Index: Case-sensitive: no
 Valexp: _____
 Valmsg: _____
 Help: _____
 Desc: _____

NextPage PrevPage **Add** Modify Delete Copy Triggers View-As
 StringAttrs GoIndex SwitchTable Browse Order Undo Exit

-----Total Fields: 0

Add New Table Data Dictionary

Database: cust1 (PROGRESS) Table: clad_mstr

Enter data or press F4 to end.

Add C1AD_EMAIL, & C1AD_EMAIL2. Both fields should have similar definitions.

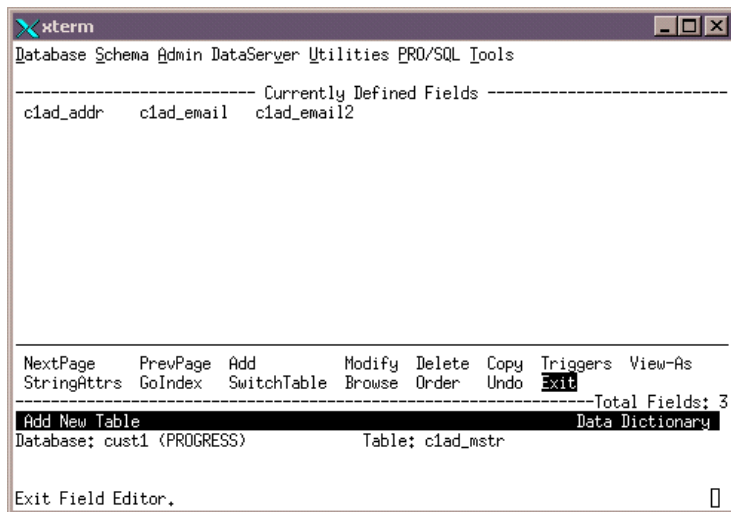
When fields are complete, press END-ERROR

QAD Proprietary

BPC-06-SCH-710

Solutions: Adding a Side Database

- ▲ Adding information to a side database
 - Automatically runs Field Editor, Add fields



```

xterm
Database Schema Admin DataServer Utilities PRO/SQL Tools

----- Currently Defined Fields -----
clad_addr  clad_email  clad_email2

NextPage  PrevPage  Add      Modify  Delete  Copy  Triggers  View-As
StringAttr GoIndex  SwitchTable Browse  Order  Undo  Exit
-----Total Fields: 3-----
Add New Table
Database: cust1 (PROGRESS)      Table: clad_mstr
Data Dictionary

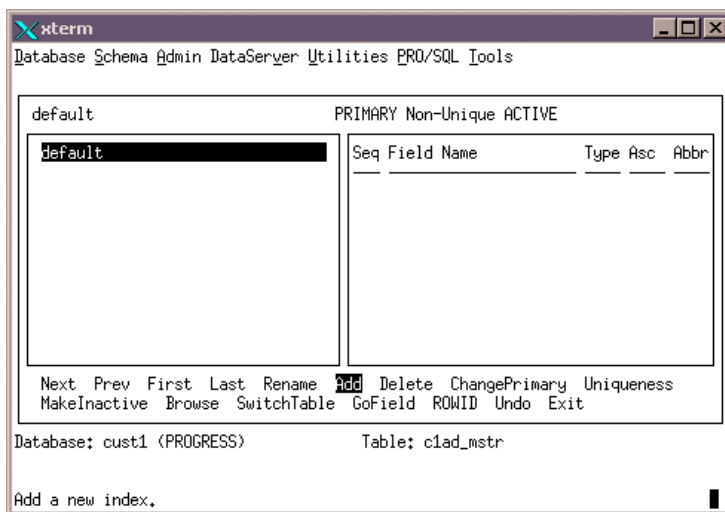
Exit Field Editor.
  
```

**Select Exit, and
apply changes**



Solutions: Adding a Side Database

- ▲ Adding information to a side database
 - Select Schema, Index Editor



**Select Add, and
create the
index
c1ad_addr**



Solutions: Adding a Side Database

- ▲ Adding information to a side database
 - Add Index, C1AD_ADDR, as Primary, Unique

xterm Database Schema Admin DataServer Utilities PRO/SQL Tools

Adding Index "c1ad_addr"

Num	Field Name	Type	Asc?
1	c1ad_addr		
2	c1ad_email		
3	c1ad_email2		

Question

Do you want this to be an ascending index component? Answering "no" will make it a descending index component.

☒ Yes ☐ No

Use the cursor-motion keys to navigate, or type field name and hilite will move.

ct fields to be nts, they are the dividing [RETURN] ields, desc, use +/- keys.

To unselect fields, move hilite above dividing line and [RETURN]. When done, press [F1].

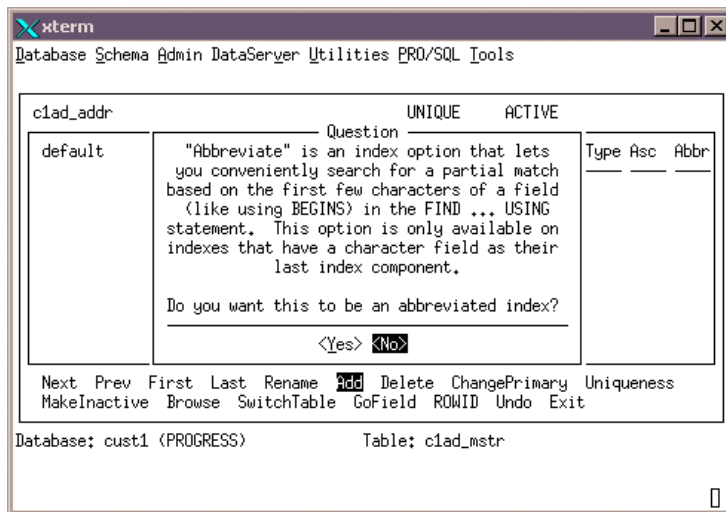
Database: cust1 (PROGRESS) Table: c1ad_

Add field C1AD_ADDR to the index in ascending order



Solutions: Adding a Side Database

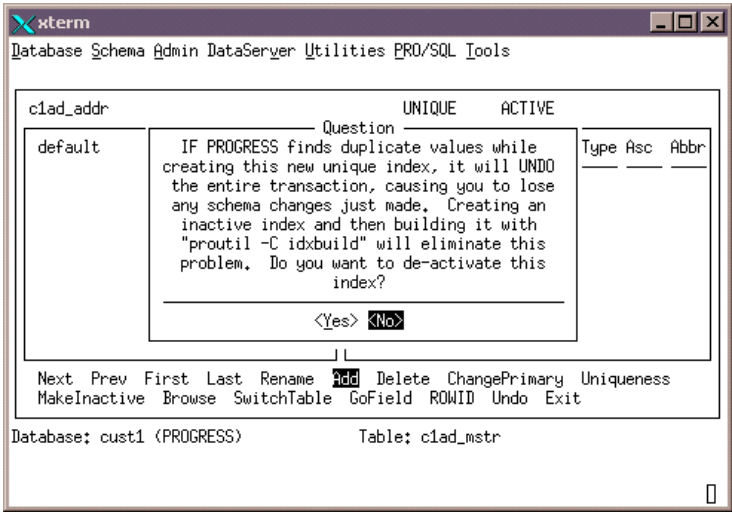
- ▲ Adding information to a side database
 - Add Index, C1AD_ADDR, as Primary, Unique



**Press GO to
add the index.
We do not
want the index
to abbreviated.**

Solutions: Adding a Side Database

- ▲ Adding information to a side database
 - Add Index, C1AD_ADDR, as Primary, Unique

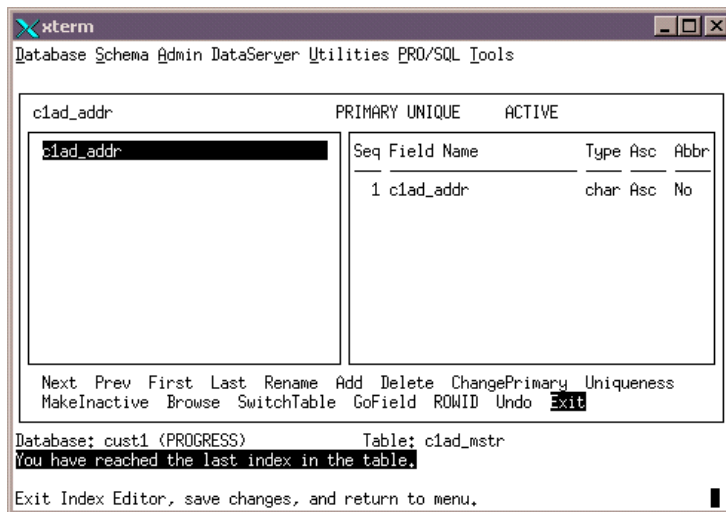


Do not "de-activate" the index when adding it.



Solutions: Adding a Side Database

- ▲ Adding information to a side database
 - Exit Index Editor



When exiting the index editor, apply the changes.

Don't forget to "freeze the C1AD_MSTR table definitions!"

Solutions: Adding a Side Database

▲ Benefits of a Side Database

- No recompiling needed
 - If you are not using the new database yet, it doesn't matter. Compile as needed
- Only need to compile those programs that are created or modified to use the new tables in the side database
- Customized r-code is easier to maintain!

Summary



Database Schema

- ✓ Identify how the database is set up
- ✓ Identify features of the database
- ✓ Freeze and unfreeze databases
- ✓ Add new schema objects
- ✓ Create a database

Chapter 7

Configuration Management

Introduction



Configuration Management

In this section, you learn:

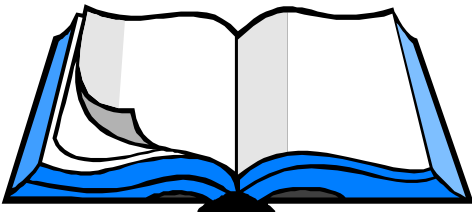
- ▲ The importance of maintaining Standard QAD Enterprise Applications
- ▲ The importance of testing in Standard, Patched and Custom code

Terminology



Terminology

▲ CVS



QAD Proprietary BPC-07-CM-024

Concurrent Versions System (CVS)

The open standard for version control, which is widely used on multi-user development projects as an additional aid to keep track of source changes and release management.

- Avoids the use of ECO patch-markers and revision history logging in software
- Used in the core application development of QAD Products

Vanilla QAD Enterprise Applications



Maintain “Vanilla” QAD Enterprise Applications

- ▲ Support issues
 - Ability to test original functionality
 - Ability to verify/pinpoint bugs
 - If you can duplicate the problem on vanilla QAD Enterprise Applications, it is easier for QAD Support to help
- ▲ Helps structure environment
 - Layer changes on top of vanilla QAD Enterprise Applications



PROPATH=.:\$MFGDIR:\$MFGDIR/us/bbi

QAD Proprietary

BPC-07-CM-034

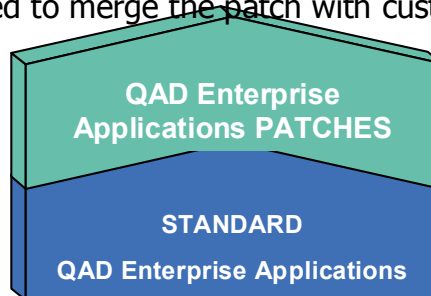
Patched QAD Enterprise Applications Code



Patched QAD Enterprise Applications Code

▲ Standard product patches

- Should be tested on top of standard QAD Enterprise Applications
- Should be moved to production but kept separate from standard QAD Enterprise Applications
 - Maintain that copy of vanilla QAD Enterprise Applications
- Evaluate the effects this has on customizations
 - Do you need to merge the patch with custom code?



QAD Proprietary

PROPATH=.:\$PRODPATCH:\$MFGDIR:\$MFGDIR/us/bbi

BPC-07-CM-049

Development Environments



Development Environments

▲ Custom code

- Should be developed in a separate environment
- Care should be taken that multiple users are not modifying the same code
 - verbal communication
 - file permissions
 - CVS
- When development is done, move the changes to a testing environment
 - Code, database changes, data, etc.



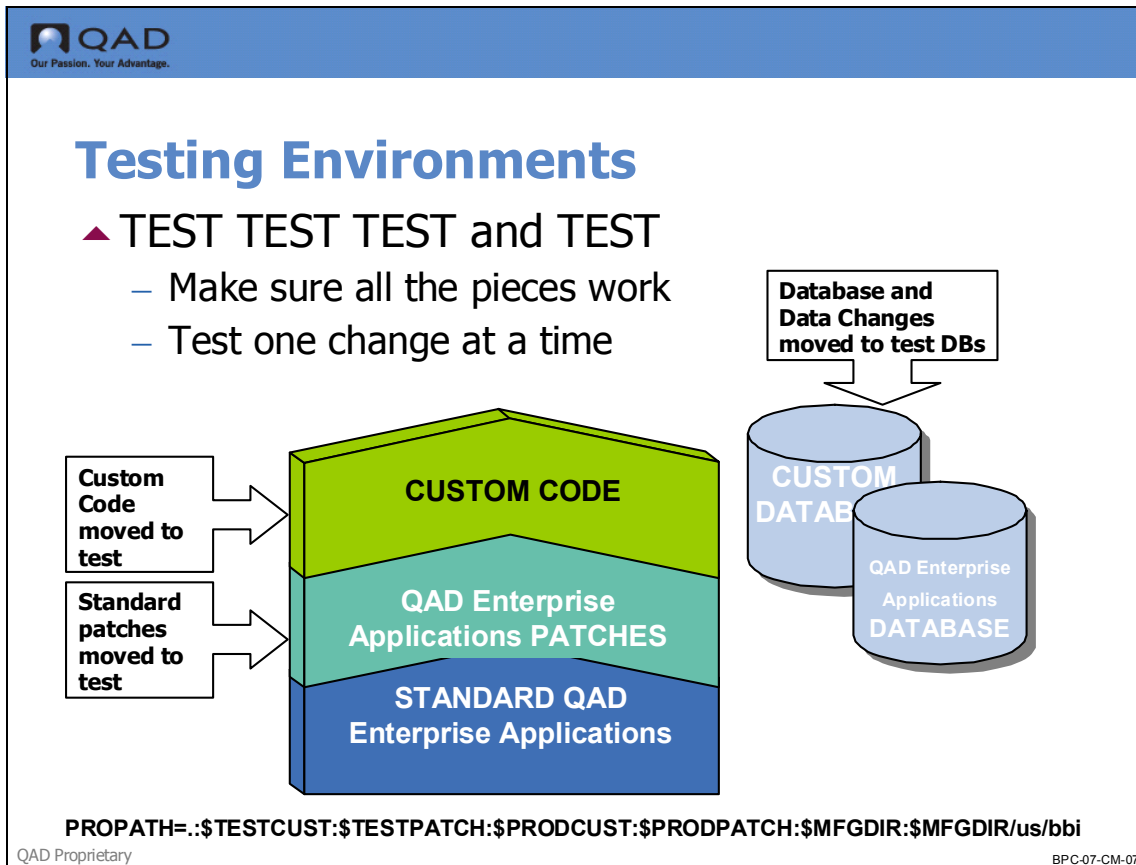
Data Management



Data Management

- ▲ Data changes need to be managed along with code and database changes
 - Browse definitions
 - Generalized Codes
 - Control Codes
- ▲ Goes through the same cycle as program changes
 - Development
 - Test
 - Production

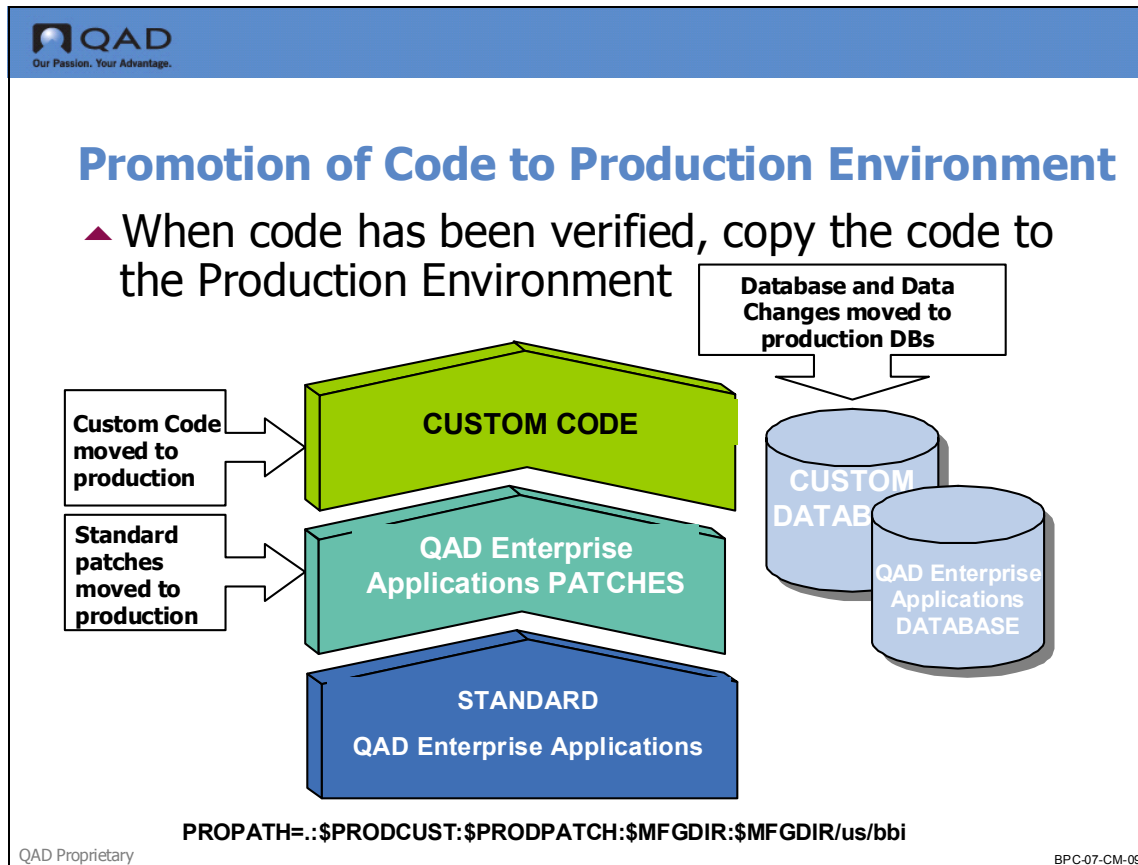
Test Environments



Things to Think About During Testing

- ▲ Program testing can be used to show the presence of bugs, but never show their absence
- ▲ Testing techniques should be such that after testing, we may have a better understanding of the product's reliability

Promotion of Code to Production



Code Promotion

- ▲ With each step of applying changes to the standard QAD Enterprise Applications
 - Keep in mind you are “promoting” the code to the next environment.
 - The changes moved to the next stage but remain in the program path
- ▲ Keeping the code separate
 - Verifying changes, bugs may require you to “peel” off the customizations and test standard QAD Enterprise Applications.
 - Perhaps you might have to “peel” off applied patches as well
- ▲ Not too separate
 - Performance problems can come from long PROPATHs

Verifying Issues



Verifying Issues

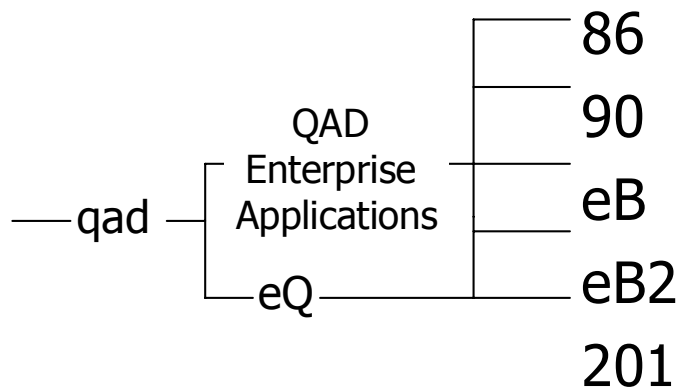
- ▲ Custom Bug?
 - Reproduce in custom environment
- ▲ Standard Bug?
 - Reproduce in standard QAD Enterprise Applications environment if possible
- ▲ Data Function?
 - Reproduce in standard QAD Enterprise Applications demo environment to eliminate data/setup errors

Managing Multiple Versions



Managing Multiple Versions of QAD Enterprise Applications and eQ

- ▲ Keep versions separate
- ▲ Directory example:



Review



Configuration Management

- ✓ The importance of maintaining Standard QAD Enterprise Applications
- ✓ The importance of testing in Standard, Patched and Custom code

Chapter 8

Programming Standards

Introduction



Programming Standards

In this section, you learn the:

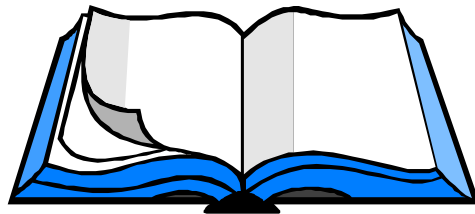
- ▲ Programming Naming Conventions
- ▲ Patch Marking Conventions
- ▲ Templates for Inquiries, Reports and Maintenance Programs
- ▲ Net UI Infrastructure
- ▲ QAD Enterprise Applications Programming considerations
(General and Desktop 2)
- ▲ QAD Enterprise Applications Translations
- ▲ Performance Considerations

Terminology



Terminology

▲ Patch / Fix / Bug / ECO



QAD Proprietary

BPC-08-PS-020

Patch/Fix/Bug/ECO

- A documented change to source code. A patch may also be referred to as an ECO, or Engineering Change Order. Patches typically refer to software fixes but could also include enhancements to functionality. Patches consist of an ECO ID along with one or more changed files. A patched file contains the ECO ID in the left margin of any line that was added or converted to a comment. No lines of code are deleted from the patched file.
- A quick-repair job for a piece of programming. During a software product's beta distribution or try-out period and later after the product is formally released, problems (called bugs) will almost invariably be found. A patch is the immediate solution that is provided to users; it can sometimes be downloaded from the software maker's web site. The patch is not necessarily the best solution for the problem and the product developers often find a better solution to provide when they package the product for its next release.
- Any instance where the code either produces an error in the data, doesn't conform to the standard look and feel of QAD's product or doesn't produce generally expected results. The R&D staff at QAD sets the standards that define a bug.
- Any software modification made to product files to change or correct functionality, appearance or performance. This change can be in response to an enhancement request or to fix a reported defect.

Naming Conventions



Program Naming Conventions



QAD Proprietary

BPC-08-PS-030

Naming Source Code

QAD
Our Passion. Your Advantage.

Naming Source Code

sosomt.p o cz
7.1 Sales Order **Maintenance** → **mt**
05/06/03

|Order:
Sold-To:
Bill-To:
Ship-To:
|

Sold-To
Ship-To

sosoiq.p a
7.2 Sales Order **Inquiry** → **iq**
05/06/03

| Item Number:
Sales Order:
Sold-To:

|Purchase Order:
Site:
Output:

sosorp.p a
7.3.1 Sales Orders by Order **Report** → **rp**
05/06/03

| Sales Order:
To:

| Sold-To:
To:

| Order Date: / /
To: / /

| Salespsn:
To:

|Purchase Order:
To:

| Action Status:
To:

| Quote:
To:

Currency (Blank for All): Base

| Include Allocated: yes
Display Features and Options: no

| Include Picked: yes
Display Comments: no

| Include Shipped: yes

| Include Unprocessed: yes
Output:

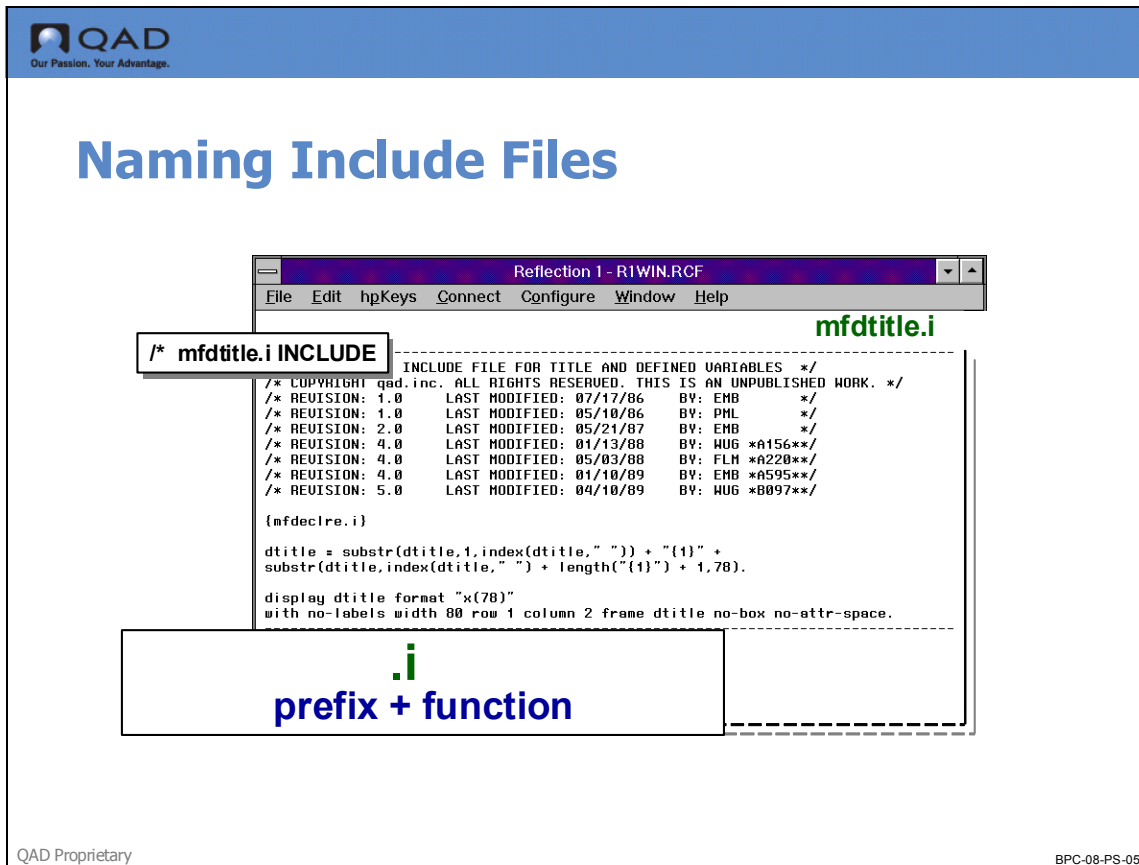
.p

module + system function + function type + sequence

QAD Proprietary
BPC-08-PS-044

The names of source code files in QAD Enterprise Applications convey information about the procedure's system module, system function, function type, and sequence.

Naming Include Files



The naming convention used for include files incorporates a two-letter prefix and a functional description into the filename. Some include files are called from procedures throughout QAD Enterprise Applications such as `gprun.i` and `mfdtitle.i`. Other include files are called only by procedures within their system module.

The general format for an include file is:

- Include filename format = prefix + function.i

The prefix for an include file could be more generic such as:

- gp for general procedure
- ic for inventory control
- mf for manager function

Naming Validation Procedures


Our Passion. Your Advantage.

Naming Validation Procedures

```

PROGRESS Data Dictionary
MODIFY-SCHEMA SQL Database Admin Utilities Reports Exit
-----
Field-Name: pod_acct      Data-Type: character
Format: x(8)             Extent:
Label: Pur Acct          Decimals: ?
Column-label: ?          Order: 160
Initial:                 Mandatory: no (Not Null)
Component of-> View: no   Case-sensitive: no
Index: no
Valexp: {gpglac.v pod_acct }
:
:
:
Valmsg: ERROR: INVALID ACCOUNT. Please re-enter.
Help:
Desc: The general ledger account to be used for PO receipts.
-----
NextPage      PrevPage      Add      Modify      Delete      Copy
GoIndex       SwitchFile    Browse   Order       Undo        Exit
-----
Database: qadddb (PROGRESS)      File: pod_det      Total Fields: 132

Press space bar to continue.

```

.v

prefix + function

QAD Proprietary
BPC-08-PS-066

All validation procedures have the file extension of .v. Validation procedures that are called throughout QAD Enterprise Applications have a two-letter prefix of gp or mf while those called by a system module will have a two-letter prefix, which is the system module ID.

Changing Standard Source Filenames



Changing Standard Source Filenames

- ▲ Renaming a customized version of a standard QAD Enterprise Applications program file
- ▲ New custom code has been created

QAD Proprietary

BPC-08-PS-071

Some factors to consider when choosing filenames for customized versions of standard code are:

- If the standard code is used by several procedures or includes within QAD Enterprise Applications, do you want all of the procedures and includes to use the customized version of the code? If you do not want all of the procedures and includes to use the customized version of the code, then you must give the customized code a filename different than the filename of the standard QAD Enterprise Applications code that you modified. The standard code will still exist with its original filename.
- If you have customized standard code and want all standard procedures to use the customized code, it is not necessary to rename the standard file. QAD recommends that you save a copy of the original code in its original state for future reference.



Customized QAD Enterprise Applications Source Filenames

Standard

.p module + function
sosomt.p

.i prefix + function
solnfrm.i

.v prefix + function
gpsite.v

Custom

.p a + n + standard filename
c1sosomt.p

.i a + n + standard filename
c1solnfrm.i

.v a + n + standard filename
c1gpsite.v

New Custom Source Filenames

.p a + n+ module + function
n1dwmt.p

.i a + n+ module + function
n1trfrm.i

.v a + n+ module + function
n1trnbr.v

Patch Marking Conventions



Patch Marking Conventions



QAD Proprietary

BPC-08-PS-100

Revision Control



Revision Control

- ▲ Engineering Change Order
- ▲ Procedure Header
(when procedure filename has changed)
- ▲ Revision Log
- ▲ Revision Number

QAD Proprietary
BPC-08-PS-110

Example:

<pre> /* c1ppptcp.p - 1.4.12 - Item Master Copy - ppptcp.p */ /* ACME GADGET COMPANY */ /* ppptcp - ITEM MASTER COPY */ * COPYRIGHT qad.inc. ALL RIGHTS RESERVED. THIS IS AN UNPUBLISHED WORK.*/ /* REVISION: 7.0 LAST MODIFIED: 06/23/92 BY: eee *F687**/ /* REVISION: 7.2I LAST MODIFIED: 07/27/93 BY: ddd *C808**/ /* REVISION: 7.2I LAST MODIFIED: 10/26/94 BY: xxx *C999**/ define variable part1 like pt_part label "Copy Item Number". define variable part2 like pt_part. /*C999* define variable yn like mfc_logical. */ ECO /*C999*/ define variable y_n like mfc_logical. ECO /*C808*/ define new shared variable xx_recno as recid. /*C808*/ define new shared variable err_flg as logical. /*DISPLAY TITLE*/ {mfdtitle.i "d*a"} /*FE43*/ /*C999/ </pre>	<pre> Procedure Header "d*a" => revision level </pre>

Engineering Change Order (ECO)

- ▲ Reflects a functional unit
 - Example: E-mail capability
- ▲ ECO number or patch number
 - QAD Enterprise Applications v6 Dnnn
 - QAD Enterprise Applications v7.3 Gnnn
 - QAD Enterprise Applications v7.4/8.4 Hnnn
 - QAD Enterprise Applications v8.5 Jnnn
 - QAD Enterprise Applications v8.6 Knnn
 - QAD Enterprise Applications v8.6E Lnnn
 - QAD Enterprise Applications v9.0 Mnnn
 - QAD Enterprise Applications v9.1(eB) Nnnn
 - QAD Enterprise Applications v9.2(eB2) Pnnn


Our Passion. Your Advantage.

Original Procedure Header

```
/* sosomt.p - SALES ORDER MAINTENANCE
/* COPYRIGHT qad.inc. ALL RIGHTS RESERVED...
/*F0PN*/ /*V8:ConvertMode=Maintenance*/
/* REVISION: 1.0      LAST MODIFIED: 03/20/86   BY: pml      */
/* REVISION: 6.0      LAST MODIFIED: 06/29/90   BY: WUG *D043*/
/* REVISION: 6.0      LAST MODIFIED: 03/20/90   BY: ftb *D007*/

.
.

/* DISPLAY TITLE */
{mfdtitle.i "c+ "}
```

Program name and description

Revision Log

Patch Numbers

QAD Proprietary

BPC-08-PS-130



Customized Procedure Header

```

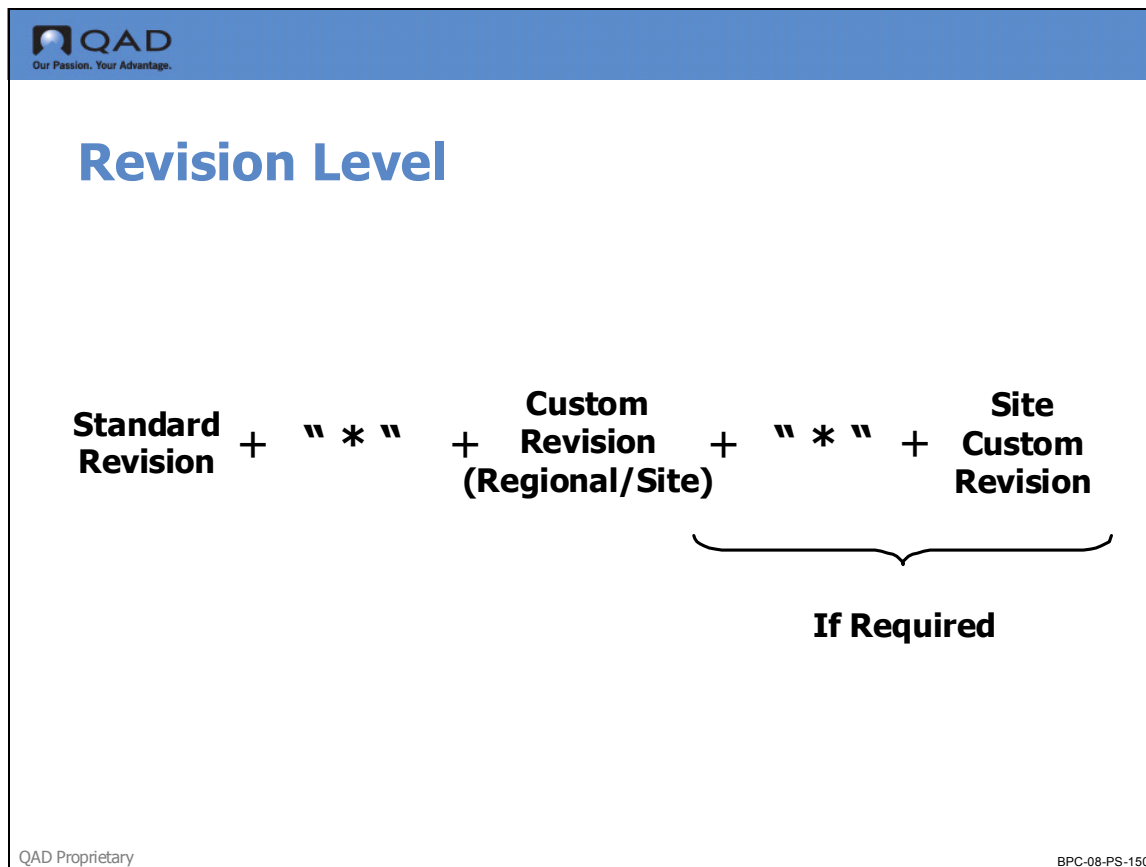
/* clsosomt.p - Custom Sales Order Maintenance      sosomt.p */
/* Custom Programs R US, USA                        Company name
New name, program
description and
original program
name

/* sosomt.p - SALES ORDER MAINTENANCE
/* COPYRIGHT qad.inc. ALL RIGHTS RESERVED...
/*F0PN*/ /*V8:ConvertMode=Maintenance*/
/* REVISION: 1.0      LAST MODIFIED: 03/20/86    BY: pml      */
/* REVISION: 6.0      LAST MODIFIED: 06/29/90    BY: WUG *D043*/
/* REVISION: 6.0      LAST MODIFIED: 03/20/90    BY: ftb *D007*/
.
.
.
/* Customization: 1.0 LAST MODIFIED: 09/25/98    BY: ame *C001*/
Revision Log

/* DISPLAY TITLE */
{mfdtitle.i "c+ "}

```

Revision Level



Revision levels identify the revisions of procedures. The revision level is found only on menu-level procedures and is displayed to the left of the menu title on the screen. Use of the mfdtitle.i include file causes both the menu title and revision level of the procedure to be displayed.

The standard format of the revision level has been expanded to allow for multiple layers of customization. The customization format for a revision level is:

```
{mfdtitle.i "aa*bb*cc"} /*dddd*/
```

where

- aa = standard procedure revision level
- bb = custom procedure revision level
- cc = site custom procedure revision level (if exists or blank)
- dddd = patch number

Inquiry Program Template



Inquiry Program Template

```
/*sampleiq.p - Sample Inquiry Program                      */
/*V8:ConvertMode= */
/*REVISION: x.x  LAST MODIFIED MM/DD/YY BY: XXX *ECOX**/

{mfdtitle.i  "g"}
REPEAT:
  [get selection criteria] WITH FRAME a.
  {mfselfprt.i "terminal" 80}
  FOR EACH [selection criteria]
    DISPLAY...
    {mfrpch.k.i}
  END.
  {mfreset.i}
  {mfmsg.i 8 1}
END.
```

Report Program Template



Report Program Template

```

/*samplerp.p - Sample Report Program                                */
/*V8:ConvertMode= FullGUIReport*/
/*REVISION: x.x  LAST MODIFIED MM/DD/YY BY: XXX *ECOX**/
{mfdtitle.i "g"}
DEFINE parameter/variable/form definitions
REPEAT:
    IF selectionx = hi_datatype_value THEN selectionx =
    default_value.
    UPDATE...
    {mfquoter.i selectionx}
    IF selectionx = default_value THEN selectionx =
    hi_datatype_value.
    {mfselbpr.i "printer" 132}
    {mfphead.i}
    FOR EACH...
        DISPLAY...
        {mfrpch.k.i}
    END.
    {mfrtrail.i}
END.

```

Report Selection Criteria

Datatype	From	To
Character	<blank>	hi_char
Date	low_date	hi_date

▲ hi_char = (depends on the character set)

ASSIGN hi_char = CHR(1).

DO i = 2 TO 255:

IF CHR(i) > hi_char THEN

ASSIGN hi_char = CHR(i).

END.

▲ hi_date = 12/31/3999

▲ low_date = 01/01/1900

hi_char is set at
start-up
depending on
what the high
character is for
the chosen
character set



Reports - Mandated Program Structure

- ▲ Every report program must:
 - Use mfrpch.k.i in all programs
 - mfrpch.k.i cannot exist within the same block level as mfrreset.k.i, mfrtrail.k.i, or mfrl080.k.i
 - If it's a Full GUI Report, then only one set/update statement (of frame a) is allowed and you MUST use 'frame a'

QAD Proprietary

BPC-08-PS-200

mfrpch.k.i cannot exist within the same block level as mfrreset.k.i, mfrtrail.k.i, or mfrl080.k.i.

- Having it on the same level would cause this include file to be skipped if an F4 is pressed or maximum pages are reached
- This include file is necessary for page scrolling and for cleaning up after the report

Reports - Mandated Program Structure

- ▲ To use the GUI converter, reports must adhere to the following:
 - DO NOT put any frame-phrase (WITH) statements on the mainloop REPEAT
 - DO NOT bundle the different sections of the report, initialize, data entry and batch quoting, with DO or REPEAT blocks
 - DO NOT use VALIDATE in the data entry statement, validate the data as a part of the post processing

QAD Proprietary

BPC-08-PS-210

In order to conform with the format expected by the GUI converter, reports must also adhere to the following.

- Do not put any frame-phrase (WITH) statements on the mainloop REPEAT
- Do not bundle the different sections of the report—initialize, data entry and batch quoting—with DO or REPEAT blocks
- Do not use VALIDATE in the data entry statement. Validate the data as a part of the post processing instead. The GUI converter cannot handle the VALIDATE statement.



Reference: Report Template & Code

- ▲ For a detailed Report template, refer to:
Appendix A: Programming Standards: Sample Report Template
- ▲ For a detailed Report code structure, refer to:
Appendix B: Programming Standards: Sample Report Program

Maintenance Program Template



Maintenance Program Template

```
/*samplemt.p - Sample Maintenance Program          */
/*V8:ConvertMode= Maintenance*/
/*REVISION: x.x  LAST MODIFIED MM/DD/YY BY: XXX *ECOX**/
{mfdtitle.I "g"}
DEFINE parameter/variable/form definitions
Mainloop: REPEAT:

    PROMPT-FOR fields... WITH EDITING:
        {mfnp.i} /*NEXT/PREV logic*/
    END.

    /* ADD/MODIFY/DELETE SECTION */

END. /* MAINLOOP */
STATUS INPUT.
```

Find an existing
record or create a
new one

Explained on next
slide



Maintenance Program Template: Add/Modify/Delete Section

```
/*ADD/MODIFY/DELETE*/
```

```
FIND table1 WHERE key1 = ... NO-LOCK NO-ERROR.
```

```
IF NOT AVAILABLE table1 THEN DO:
```

```
{mfmsg.i 1 1} /* ADDING NEW RECORD */
```

```
Create New Record.
```

```
END.
```

```
Ststatus = stline[2].
```

```
STATUS INPUT ststatus.
```

```
DISPLAY fields.
```

```
Setloop: ON ENDKEY UNDO blockx, RETRY blocky:
```

```
SET fields... GO-ON (F5 CTRL-D).
```

```
If LASTKEY = KEYCODE(F5) OR ... /*DELETE*/:
```

```
del_yn = YES.
```

```
{mfmsg01.i 11 1 del_yn}
```

```
IF del_yn = YES THEN DO:
```

```
DELETE table1. NEXT mainloop.
```

```
END.
```

```
END.
```

```
VALIDATIONS ...
```

```
END. /* SETLOOP */
```

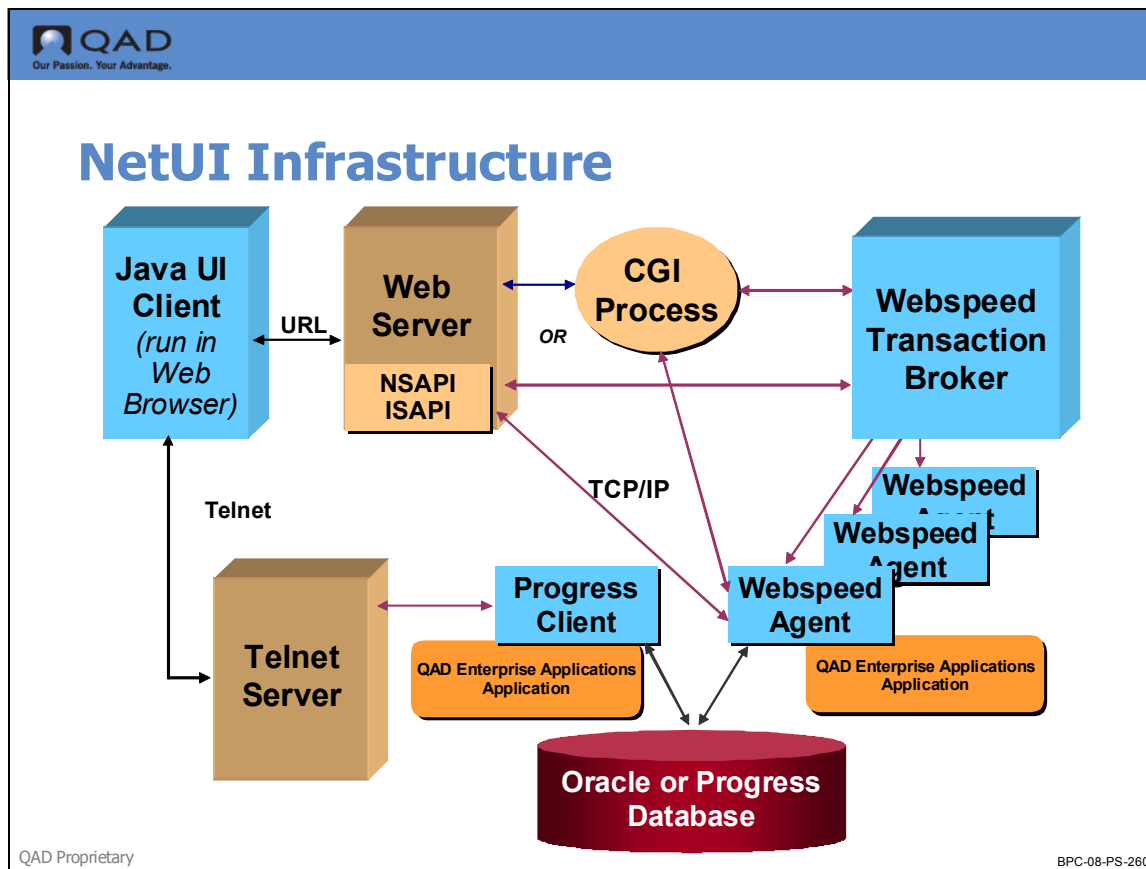
If the record is not
found, create it!

Update status
line and display
fields

Maintenance Programs - Constraints

- ▲ If you have to print from a maintenance program, put all the output commands in a subprogram
- ▲ Use WIDTH 80 when defining frames in maintenance programs
- ▲ Do not use explicit WITH ROW <line-number> where the line-number is hard-coded. NEVER place frames above line 3!
- ▲ All Fields in an input frame must be defined within a single FORM statement

Net UI Infrastructure



Java UI



JAVA UI

- ▲ This section is not a HOW TO (at this point) but is meant to make you aware of what you will see in the code
- ▲ Some reports and inquiries have been “web enabled”
 - New include files
 - New procedures
 - Coding practices
 - Tags

JAVA UI New Include Files

Used in	Name	Description
Browses	wbbr01.i	Inserted by the browse generator so that all browses/lookups are automatically web-enabled. Instantiates the persistent library routine wbbr01.p
Reports	wbrp01.i	Placed in the menu-level program before any blocks and after the form statements. Instantiates the persistent library routine wbrp01.p
Reports	wbrp02.i	Placed in subprograms that must be web-enabled before any blocks and after the form statements. Verifies the persistent library exists and establishes shared variables
Reports	wbrp06.i	Placed before the interactive statement (prompt-for,update,set). Initializes the input frame to the JAVA UI. Accepts the input back from the UI and repopulates the screen values

JAVA UI New Procedure Files

Used in	Name	Description
Browses	wbbr01.p	Persistent library routine that is launched by wbbr01.i and handles all communications with the JAVA client through established protocols
Browses	wbqu01.p wbqu02.p wbqu03.p	Run-time compiled procedures that perform the "open query" and return records for the result set
Reports	wbrp01.p	Persistent library routine that is launched by wbrp01.i or wbrp02.i and handles all communication with the JAVA client through established protocols



JAVA UI Sample Inquiry Program

```
/* Sample Inquiry Program */
```

```
{wbrp01.i}
```

```
REPEAT:
```

```
IF c-application-mode <> "web":U THEN
```

```
  [Get selection criteria (update, set or prompt-for)]
```

```
  {wbrp06.i &command=[update,set, or prompt-for]
```

```
    &fields="[Selection criteria]"
```

```
    &frm="[Frame name]"}
```

```
{mfselfprt.i terminal 80}
```

```
FOR EACH...
```

```
  DISPLAY...
```

```
  {mfrpch.k.i}
```

```
END.
```

```
{mfreset.i}
```

```
END.
```

Every inquiry program must follow this structure

JAVA UI Sample Report Program

```
/* Sample Report Program */
```

```
REPEAT:
```

```
IF c-application-mode <> "web":U THEN
```

```
  [Get selection criteria (update, set or prompt-for)]
```

```
  {wbrp06.i &command=[update,set, or prompt-for]
    &fields="[Selection criteria separated by a space]"
    &frm="[Frame name]"}
```

```
  [Assign variable from selection criteria]
```

```
  {mfquoter.i [selection criteria]} /*if batch runnable*/
```

```
  {mfselfprt.i terminal 132}
```

```
  {mfthead.i}
```

```
FOR EACH ...
```

```
  DISPLAY ...
```

```
    {mfrpch.k.i}
```

```
END.
```

```
  {mfrtrail.i}
```

```
END.
```

Every report program must follow this structure



Another JAVA UI Sample Report

```

/* Sample Inquiry Program */
{wbrp01.i}
REPEAT:
  IF c-application-mode <> "web":U THEN
    [Get selection criteria(update, set or prompt-for)]
    {wbrp06.i &command=[update,set, or prompt-for]
      &fields="[Selection criteria]"
      &frm="[Frame name]"}
  IF (c-application-mode <> 'web':u) OR
  (c-application-mode = 'web':u AND
  (c-web-request begins 'data':u)) THEN DO:
    [Assign variable from selection criteria]
    {mfquoter.i [selection criteria]}
  END.
  {mfselbpr.i terminal 132}
  {mfphhead.i}

  FOR EACH...
    DISPLAY...
    {mfrpchkh.i}
  END.
  {mfreset.i}
END.

```

After frame input, setup the input variables to run report!

JAVA UI Other Changes in the Code

▲ New Tags in the code

(used by QAD Development)

- /*RunMode=Web*/
 - Programs that are referenced only in the web environment
- /*WebEnabled=No*/
 - Programs (menu level and subprograms) that are not being used in the web production structure

▲ New Table (gui database)

- pgmi_mstr - Program Information Master
 - Used in conjunction with the new tags



Java UI Report Issues

- ▲ Shared Frames
 - modify the menu-level program to act as a “wrapper” and move all functionality into a new subprogram
- ▲ Using the same variable in both input and output frames can cause the value to not be set correctly
 - Add another variable
- ▲ Placement of wbrp06.i is critical
 - Always place wbrp06.I before the first interaction statement

Java UI Report Issues

- ▲ PROGRESS errors can cause output not to display
 - Make sure all keywords that can generate a run-time error have the “no-error” attribute and are handled appropriately
- ▲ Updates to the input frame must come before the “output selection” include file
- ▲ Use Run-time conditions to avoid
 - Statements that “block” for user interaction
 - pause
 - message
 - Keywords that cause the report to end or to restart
 - return
 - leave



Java UI Report Issues

- ▲ Run-time conditional statement
 - if c-application-mode <> "WEB" then
 - conditional cannot be a "DO" block
- ▲ Use Run-time conditions to avoid statements that "block" for user interaction
 - pause
 - message
- ▲ Keywords that cause the report to end or to iterate/restart must cause the the report to fall through to the end



Reference – NetUI Coding Guidelines

For more details on NetUI Coding guidelines, refer to:

Appendix C: How to Web-Enable a Report

Programming Considerations



General QAD Enterprise Applications

- Database Tables
- Database Fields

Database Tables



Database Tables

- ▲ Only 254 fields per table are allowed in QAD Enterprise Applications eB and below; 1000 fields per table are allowed in QAD Enterprise Applications eB2 and above.
- ▲ Table names are a maximum of 14 characters long and consist of a prefix, an underscore, and a suffix.
- ▲ Table names should be entered in lower-case only.
- ▲ Table name must be unique within the first 8 characters for purposes of dumps and loads.
- ▲ Use TEMP-TABLES instead of WORKFILES
 - TEMP-TABLE is initiated from scratch each time it's used so you can be sure that it is always empty.
- ▲ The **qad_wkfl** is reserved for QAD use and it is **not** to be used by customers to store customer-specific data

QAD Proprietary

BPC-08-PS-396

The work files in the database have been used to temporarily store data within a program or a set of programs and sub-programs. It is not used to persist any data that other users can access; qad_wkfl is an exception.

The use of database tables for temporary data creates the problem that there should not be any data when you begin your transaction that could mess up the processing in a program. To ensure that the xx_wkfl is empty, you have to delete any records before processing starts. This creates an unnecessary source of error and extra complexity in each program using xx_wkfl.

Database Fields



Database Fields

- ▲ Give Field Labels a Full & Complete name
- ▲ New field validations should be implemented in procedural code not in schema validations
- ▲ NEVER store RECID as a foreign key to another record
 - Use sequences instead of recid's if you need to have a unique identifier for records that otherwise are not unique (such as tr_hist and exru_usage)

QAD Proprietary

BPC-08-PS-400

Field Validation versus Procedural Code Validation

In accordance with standards prior to the release of QAD Enterprise Applications eB, the preferred approach to field validation was to use schema validations. However, with the advent of XUI, it became necessary to implement field validations in procedural code. This has created a situation where the same functionality exists in schema validations to support CHUI/GUI and in procedural code (in ROPs) to support XUI. For example, schema validation **gpcode.v** performs the same function for

CHUI/GUI as does `validateGeneralizedCodes` in **gpcodxr.p** for XUI. To avoid any future duplication of functionality, all field validation required in future development activities should be implemented in procedural code rather than in schema validations. This functionality will then be shared by both CHUI/GUI and XUI. Additionally, the implementation of validation logic in procedural code accomplishes the desirable separation of business logic from the UI and the database.



RECID Example

Right:

```
define variable tr-recno as recid no-undo.
create tr_hist.
{mfrnseq.i tr_hist tr_hist.tr_trnbr tr_sq01}
tr-recno = recid(tr_hist).
```

/ OK to store recid in a variable to use in same session */*

for first tr_hist where recid(tr_hist) = tr-recno no-lock: end.

Wrong:

```
create tr_hist.
tr_trnbr = recid(tr_hist).    /* Should not store a recid */
```

/ No guarantee that if a dump/reload occurs, recid will not be reused, causing duplicate key error */*

QAD Proprietary

BPC-08-PS-410

RECID

- Do not use a stored recid as a foreign key to another record
- Recids are changed when doing a dump and load of the database as the recids are rebuilt
- Update the program “mgsqmt09.p” with information about the new sequence to allow for future maintenance
- Update the sequence update utility, utsequp.p and utsequp2.p, with the new sequence



Mandatory Fields

- ▲ When creating new fields in the schema, make them “Mandatory” unless it is a Date field
- ▲ Field names are limited to 22 characters and must be named and labeled appropriately
 - They must be in lower-case only and begin with the table prefix, e.g., so_nbr for so_mstr
 - Name your fields consistently across tables, particularly those that are used for joining
 - Do not give similar field names to fields that are not similar
 - If two fields are similar, (e.g. jcd_surf_spot, wug_surf_spot), then the labels should be the same, “Favorite Surf Spot”
 - Leave column labels undefined (?) if they are the same as the side label
- ▲ Do not use a format length greater than 255 for a character field

QAD Proprietary

BPC-08-PS-420

Mandatory Fields

When creating new fields, either on new tables or existing ones, make them Mandatory unless you absolutely must allow an unknown value in the field. When fields are made mandatory, character fields can still be left blank and numeric fields can still be left zero, they just cannot contain the unknown value.

The use of the Mandatory flag does NOT mean that the user must enter something other than blank in the field. The validation phrase must be used to check for blank values regardless of the mandatory setting.

If a mandatory field is entered by the user via an input field, the program should explicitly check for the unknown values (?) and issue an appropriate error message.

Database Indexes



Database Indexes

- ▲ Every table must have a unique primary index.
 - The unique index should consist of the minimum number of keys to guarantee uniqueness of a record in a table
- ▲ Always try to select records where the leading fields of an index are set (via=) to a single value.



PROGRESS Keywords

- ▲ Do not abbreviate
 - New versions of PROGRESS could mean new keywords
 - A unique abbreviation of a keyword in one release does not mean it will be unique in the next
- ▲ Do:
`define variable mychar as character.`
- ▲ Don't:
`def var mychar as char.`



QAD Enterprise Applications Variable **mfguser**

- ▲ The variable **mfguser** should always be set by initialization during the QAD Enterprise Applications login process
- ▲ Use it as a unique value across sessions
- ▲ Do not assign it a new value

▲ Do:

```
create usrw_wkfl.
assign usrw_key1 = mfguser
        usrw_key2 = counter.
```

▲ Don't:

```
assign mfguser = usrw_key1.
```

QAD Proprietary

BPC-08-PS-450

- The variable **mfguser** is a global variable that contains a session identifier that is unique among all sessions logged on at any one time
- The variable **mfguser** is used when temporary records are created that need to be uniquely identified for a particular QAD Enterprise Applications session, typically work-file records
- **mfguser** is a shared variable declared in `mfdeclre.i` and is set by `mfla.p` upon initial entry into the QAD Enterprise Applications main menu



Accessing _File, _Field, and _Sequence Attributes

- ▲ Do not directly access the meta schema
 - PROGRESS only searches the first connected database
- ▲ gpfile.i, gpfield.i, gpseq.i
 - Searches ALL the connected databases to get the attributes of a given schema object

▲ Do:

```
{gpfilev.i} /*define needed variables*/
{gpfile.i &file_name=so_mstr}
if file_found then ...
```

Available Information:

file_found
file_desc
file_valexp
file_valmsg
file_frozen
file_dump_name

▲ Don't:

```
find _file WHERE _file-name = "so_mstr".
if available _file then ...
```

PAUSE 0 Statement

- ▲ Avoid using the BEFORE-HIDE option.

Pause 0 is preferred

- PAUSE 0 BEFORE-HIDE sets a global PROGRESS condition that causes problems for field and procedure help, and multi-page inquiries that rely on the default behavior of hiding frames

- ▲ Do:

`pause 0.`

- ▲ Don't:

`pause 0 before-hide.`



Specify Lock Status

- ▲ NO-LOCK
 - Record reads (report and inquiry programs)
- ▲ EXCLUSIVE-LOCK
 - Record updates (maintenance programs)
- ▲ Avoid SHARE-LOCKS - can cause deadly embraces
- ▲ Do:

```
for each so_mstr where so_nbr = nbr no-lock:
    display so_nbr.
```
- ▲ Don't:

```
for each so_mstr where so_nbr = nbr:
    display so_nbr.
```

QAD Proprietary

BPC-08-PS-480

Share-locks are what you get by default when you do not specify the type of lock.

FIND... NO-ERROR

- ▲ When performing a "find" statement, use the "no-error" clause and then test for availability.
- ▲ Do your own error checking
- ▲ Avoid displaying PROGRESS error messages


```

      ** FIND FIRST/LAST failed for table table_name.
      (565)"
      
```
- ▲ Do:


```

      ** FIND FIRST/LAST failed for table so_mstr. (565)
      find first so_mstr where so_nbr = nbr no-lock no-
      error.
      if available so_mstr then
        display so_nbr.
      
```
- ▲ Don't:


```

      find first so_mstr where so_nbr = nbr no-lock.
      display so_nbr.
      
```

QAD Proprietary

BPC-08-PS-490

When performing a Find statement, use the "no-error" attribute, which suppresses the Progress error message when the record is not available.

When appropriate, such as when there is a change that the record does not exist, display an appropriate message and undo, retry, leave, and so on.

mfdttitle.i and mfdeclre.i

- ▲ When used, {mfdttitle.i} or {mfdeclre.i} must directly follow the source file header.
 - The first executable (non-comment) in the source file

▲ Do:

```

/* sosomtb.p - SALES ORDER HEADER TAX DATA
 */
/* COPYRIGHT qad.inc. ALL RIGHTS RESERVED. THIS IS AN UNPUBLISHED
WORK.
 */
/*H0CJ*/ /*V8:ConvertMode=Maintenance
 */
/*K1Q4*/ /*V8:WebEnabled=No
 */
/* REVISION: 7.3      LAST MODIFIED: 02/04/93   by: bcm *G415*
 */
/* REVISION: 7.4      LAST MODIFIED: 06/24/93   by: pcd *H008*
 */

```

{mfdeclre.i}

Include mfdeclre.i at the top of the variable definitions

```

define shared variable so_recno      as recid.
define shared variable new_order     like mfc_logical.
define shared variable undō_sosomtb like mfc_logical.

```

Data Entry Frames

- ▲ Always specify a WIDTH
 - WIDTH 80 for full screen data entry frames
 - Do not rely on default frame sizing
 - OVERLAY frames do not need a WIDTH specification
- ▲ Use SKIP to break a line in the frame
 - Do not rely on the field fit in a frame
 - GUI might surprise you
- ▲ Define all fields within a FORM statement
 - Do not add fields via a DISPLAY, UPDATE, SET, etc. command
 - The only exception is the "Output" and "Batch ID" fields added to FRAME A by the mfselbpr.i and mfselptr.i include files
- ▲ All displayed frames should have setFrameLabels() function called after frame definition.
- ▲ Enclose the WHEN clause conditional within parenthesis

QAD Proprietary

BPC-08-PS-510

setFrameLabels() function should not be called in the following situations:

- 1 Frame is defined as no-labels frame
- 2 All fields in the frame are defined as no-labels
- 3 Frame is defined as header



Message Numbers

- ▲ Numbers 9000 to 9999 are supposed to be reserved for customers but they are not
- ▲ Numbers 10000 to 19999 are reserved for customers and distributors but you need to do some customizations to the message number format in the following programs:
 - Message Maintenance
 - Message Report
- ▲ Custom Messages should be created within this range to ensure that minor and major upgrades will not use the same message numbers
- ▲ Never change the context of a message or delete an existing message

System Data

- ▲ If there is required system data:
 - Include a routine that will create the required data if it does not exist in the database
 - Protect installation errors
 - System will be automatically restored in the event that the system data is inadvertently deleted



Local Variable Naming

- ▲ When selecting names for local variables, use enough description to distinguish the field from similarly named fields in the software
 - Prevents problems occurring with
 - field/procedure help
 - scrolling windows
 - cross-referencing
- ▲ Be careful not to name the variable like a schema field unless you are specifically mapping that field to a `xx__qad##` field and plan to add that field to the table in the next release
- ▲ Local variables representing a particular item should be named consistently throughout the system

QAD Proprietary

BPC-08-PS-540

Use meaningful and descriptive names for local variables

- Consider the variable “line.” In QAD Enterprise Applications, this variable could mean order line, comment line, page line number, and so on. Use a self-explanatory name, albeit longer, to distinguish the variable: `sales_line`, `cmmt_line`, `line_nbr`.

Local variables should be named consistently

- For example, `tax_class`, a local variable, which could be used as selection criteria on a report, a temporary data holder in a process, and so on, should always be named exactly the same between programs. This simplifies the process of code modification, especially where data-entry and, therefore, field help, scrolling windows, and so forth are concerned.

Local Variable Naming

- ▲ Variable names to be careful when using:
 - acct, addr, cc, cd, class, code, date, desc, id, line, list, nbr, number, part, qty, ref, time, tot, trl, type, weight, wt
- ▲ Do not use upper-case letters in variable names. Upper-case letters are typically not found if someone is doing a grep on the code looking for a string.



Local Variable Naming

Correct:

Example 1:

```
define variable pt-head as character no-undo.  
define variable pt_kickback as character no-undo.
```

for first pt_mstr exclusive-lock:

```
    pt_kickback = pt__qad13.  
    /* You must plan to rename pt__qad13 to pt_kickback in the next release!  
    */  
end.
```

Example 2:

```
define variable number-lines-on-order as integer  
no-undo.
```

Local Variable Naming

Wrong:

```
define variable pt_head as character no-undo.  
/* Variable named as if it was a database field. */
```

```
define variable NumberLinesOnOrder as integer  
no-undo.  
/* Upper and lower case mixed */
```



Definition and Acceptable Values for Logical Variables

- ▲ The initial value of logical field should always be **yes** or **no** in lower case
- ▲ Use “like mfc_logical” if the variable might be used in UI

QAD Proprietary

BPC-08-PS-580

Use of “like mfc_logical” when defining the variable ensures the consistency of field attributes and column labels.



Definition and Acceptable Values for Logical Variables

Correct:

```
define variable valid_acct  like mfc_logical  
                           initial yes no-undo.  
define variable summary    like mfc_logical  
                           format "Summary/Detail"  
                           initial yes no-undo.
```

Note: "Summary/Detail" should be a preprocessor label
rather than a hard coded value



Incorrect Definition for Logical Variables

Wrong:

```
define variable valid_acct like mfc_logical  
        initial "yes" no-undo.
```

Note: "yes" would be translated

```
define variable summary like mfc_logical  
        format "Summary/Detail"  
        initial "Summary" no-undo.
```

Note: "Summary" would need to be translated.

Appropriate Use of Define...Like

- ▲ Use DEFINE....LIKE syntax only to define new variables that will represent the same kind of data as the field referenced by the "like" keyword
- ▲ Always define COLUMN-LABEL for the variable if your program uses a column label for it

QAD Proprietary

BPC-08-PS-610

- 1 Using “define...like” as a shorthand for creating new variables causes coupling between the variable being defined and the referenced field. This coupling can lead to an unintended change in the variable when the field definition is changed. Also the database containing the referenced field must be connected at both compile time and run time.
- 2 In most cases, the field referenced by the “like” keyword will have a missing value for its column label, so the new variable will have a missing column label as well. Progress’ default action when printing is to use the field label as the column label if no column label is explicitly defined. In such a case, the field label would print as the column label for the new variable. This leads to translation problems: Because words in English are shorter than in other languages, translated labels are often too long to be used as column labels. In order to make the translated labels fit the space allocated in a form, the labels are heavily abbreviated and added to the (translated) schema as column labels. The problem occurs when a variable that is defined to be LIKE another field has a new column label defined for it. Even though the new field may have a new label assigned to it in the definition statement, the new variable will inherit the column label from the source field. Of course this (incorrect) column label will print whenever a column label is called for. This type of error can be hard to spot and it adds complexity and cost to the translation process.



Procedure Structure

- ▲ Use an include file to define the shared Progress elements.
 - By keeping shared elements in one physical file, code maintenance can be simplified
 - Use NEW SHARED in the top-level program
 - Use SHARED in the lower-level programs
 - Pass the NEW and SHARED as arguments to the include file
 - If the Progress elements are not shared across multiple programs, include the file without any arguments
 - Comments are essential to explain the purpose of the include file
 - Include file should have the standard header regarding copyright message, purpose, and so on

Note: An example of and more details on Shared Variable usage is covered under Performance Considerations later in the chapter

QAD Proprietary

BPC-08-PS-620

- If shared Progress elements, such as frame and input/output stream, need to be used in more than one program, then they must be defined as shared. In Progress, frames and streams cannot be sent in as input parameter to another program/procedure.
- To avoid duplicating code about the shared elements in multiple programs, use an include file to define the shared frames, streams or any other Progress constructs that cannot be sent as parameters
- At the top-level program, use NEW SHARED as the argument for the include file and in lower-level programs include the file with SHARED as the argument
- If the Progress elements are not shared across multiple programs but simply re-used, include the file without any argument
- Make sure that there is a comment line above the include file explaining the purpose of the include file
- Also make sure that the include file has the standard header regarding copyright message, purpose and so on
- By keeping shared elements in one physical file, code maintenance can be simplified

Procedure Structure

incfile.i

```
/* header information */
define {1} frame a.
define {1} stream rpt.
form
    field1 field2 with frame a.
```

program1.p

```
/* program1.p - Main program */
define variable error_flag as log.
/* This include defines shared variables and shared
frames */
{ incfile.i "new shared" }
    <more code here>
{gprun.i "" program2.p"" "(output error_flag)"
```



Procedure Structure

program2.p

/* program2.p - Subprogram */

{ incfile.i "shared" }

Procedure Structure

- ▲ Use of the DISPLAY and SET statements is preferable to the UPDATE statement in blocks with UNDO conditions
- ▲ **Avoid using shared variables**
- ▲ Never introduce a shared variable in a new program/procedure
- ▲ Pass the variable as an input or input-output parameter.
- ▲ Do not send the recid of a record to another procedure to read it again. Instead, send a buffer as a parameter.
- ▲ Always fully identify the type of end statement when using PROCEDURE, CASE, and FUNCTION.

Note: Refer to Performance Considerations covered later in the chapter on Buffers

QAD Proprietary

BPC-08-PS-650

When UPDATE is used within a block with UNDO condition and the UNDO is invoked, all fields are reset to their original values.

When the values are DISPLAYed outside a block with UNDO condition and then a SET is used inside the block, the fields retain their edited values.

Exceptions:

- System-level shared variables (for example, user-id and password) and application-level shared variables (for example, part, site, and entity) are exceptions to the standard.
- Split-reports and inquiry programs at the top level. It is okay to define selection-criteria variables at the first-level split-programs as shared variables.
- Programs where shared variables are already used extensively across multiple programs.

Although Progress accepts an END keyword by itself for PROCEDURE, CASE, and FUNCTION blocks, always fully identify the END statement. Additionally, PROCEDURE, END PROCEDURE, CASE, END CASE, FUNCTION, and END FUNCTION statements should always in be upper case.

Procedure Structure

- ▲ Other end statements benefit from having a comment after them to identify which code initiated the block

do while true:

 <code>

end. /* do while */

Status Line

- ▲ Present uniform function key usage to the user
- ▲ Use the status lines contained in `stline[ ]`
 - `stline[ ]` is a global variable defined in `mfdeclre.i`
 - The values of `stline` are set in `mf1a.p`
 - Restricted to 63 characters



Status Line

▲ Do:

- ststatus = stline[1].
- status input = ststatus.

▲ Don't:

- status input off.
- status default "F1=Go F2=Help F3=Ins F4=End" + "F6=Menu F7=Recall F8=Clear F9=Prev F10=Next".

Status Line

- ▲ stline[1] is used with the mfnp#.i include files
- ▲ stline[2] & stline[3] are used in maintenance programs
- ▲ stline[10] is used in sosgadup.p
- ▲ stline[12] is used in apvomtc1.p
- ▲ stline[13] is used in rcctwbb.p, rcmbmta.p and rcshwbb.p
- ▲ stline[4], stline[5] and stline[6] were used in the old scrolling windows (there are some of these still in the product)



GUI Converter

- ▲ The Converter is a C program that functions as a PROGRESS version 6 tokenizer that inspects and modifies source code
 - It parses PROGRESS v6 programs to the QAD Enterprise Applications version 7 standards, restructures the programs, and adds PROGRESS v7 language elements so that the programs look good when running under WINDOWS
- ▲ The Converter is the standard method for implementing Windows versions of QAD Enterprise Applications character based products

GUI Converter



GUI Converter

For each QAD Enterprise Applications release, there's a related version of the GUI converter (or check previous converted program):

- QAD Enterprise Applications 7.3 Version 1.60 & 1.66
- QAD Enterprise Applications 8.5D Version 1.69
- QAD Enterprise Applications 8.6E Version 1.71
- QAD Enterprise Applications 9.0 Version 1.71
- QAD Enterprise Applications eB Version 1.76



The GUI Converter Executable

```
convert [-m|r|g|p|d|f] [-v] [-notwinkle] [-ini=iniFile]
      Filename ...
```

```
convert [-M|R|G|P|D|F|L] [-v] [-notwinkle] [-ini=iniFile]
      [Listfile]
```

Opt	Listfile	File	Description
M	mt.wrk	status.mt	convert a maintenance program
R	rp.wrk	status.rp	convert to a minimal GUI report program
G	rpmain.wrk	status.gr	convert to a standard GUI report program
D	src.wrk	status.cnv	convert in Declare-move mode
F	src.wrk	status.cnv	convert in Field-list mode
L	src.wrk	status.cnv	look in listfile for input filenames
P	src.wrk	status.cnv	generate "pretty print" output
v	N/A		verbose mode
notwinkle			suppress "still working" indicator
ini=iniFile			Specify alternate convert.ini file

The convert.ini File

- ▲ Source directories
 - Source files will be read from the directories in the comma delimited search path "srcSearchPath=". Must have at least one valid directory
- ▲ Target directory
 - Output files are written to "targetDir=". The output filename is the same as the input file. Valid, non-blank target dir. is required
- ▲ Work directory
 - Listfiles will be read from "workDir=". The work directory defaults to the local directory if left blank
- ▲ Overwrite behavior (optional)
 - Putting the "overwriteExistingTarget=YES/NO" line in the ini file is optional. The default is YES, allowing files to be overwritten

convert.ini

```
overwriteExistingTarget=YES  
srcSearchPath=/users/ame/work/ptchsrc,  
/users/ame/work/newsrc,/users/ame/work  
/src  
targetDir=/users/ame/work/converted  
workDir=/users/ame/work
```

Convert Command Line

```
convert -v -L mylist.lst
```

mylist.lst

```
c1sosomt.p  
c1sosomta.p  
c1ppptmt.p  
.  
.  
.
```



Converter Tokens (1 of 10)

`/*v8! ... */`

- ▲ The ConvertMode token is used by the GUI Converter to modify the source file to run in Windows.
- ▲ Used to enable code for QAD Enterprise Applications v8.x
- ▲ The ConvertMode token is always placed in a comment statement after the Revision History header section.
- ▲ Converter removes the leading and trailing comments and then converts the previously enclosed code

Before convert:

```
message "Appear in alert-box for GUI" /*v8! VIEW-AS ALERT-BOX  
*/.
```

After convert:

```
message "Appear in alert-box for GUI" view-as alert-box.
```


Converter Tokens (2 of 10)

`/*V8-*/` ... `/*V8+*/`

- ▲ This is used to omit code from the converted version of the source file
- ▲ Code found between these tokens is not processed by the converter and not written to the output file

Before convert:

```
message "Do not pause in GUI after this message" /*V8-*/
PAUSE. /*V8+*/
```

After convert:

```
message "Do not pause in GUI after this message"
```



Converter Tokens (3 of 10)

/*V8:ConvertMode=Maintenance*/

- ▲ This directive tells the GUI converter to treat the source file as a maintenance program
- ▲ No report output is generated
- ▲ All frames will have attributes appropriate for screen display
- ▲ There can be several levels of nested transactions

Converter Tokens (4 of 10)

```
/* sosomt.p - Sales Order Maintenance */
/* Copyright 1986-2008 QAD Inc., Carpinteria, CA, USA. */
/* All rights reserved worldwide. This is an unpublished work. */
/* */
/* Revision: 1.0 BY: Joe Coder DATE: 01/10/00 ECO: YYYY */
/*-End of revision history----- */
/* */
/*V8:ConvertMode=Maintenance */
```

Converter Tokens (5 of 10)

/*V8:ConvertMode=FullGUIReport */

- ▲ Should be used for all new reports
- ▲ There will be one frame, named frame a, that accepts user input and will have screen display attributes. All other frames will be used for hard copy output
- ▲ Only one REPEAT block is expected to be used for generating the output
- ▲ The output source program to be generated will fit with the event-driven user interface model and look more "GUI"
- ▲ All FullGUI reports have a print button under the report parameters that produce a dialog-box to choose the output destination

Converter Tokens (6 of 10)

```
/* wowsrp01.p - Work Order component shortage by part report */
/* Copyright 1986-2008 QAD Inc., Carpinteria, CA, USA. */
/* All rights reserved worldwide. This is an unpublished work. */
/* */
/* Revision: 1.0 BY: Joe Coder DATE: 01/10/00 ECO: *YYYY* */
/*-End of revision history----- */
/* */
/*V8:ConvertMode=FullGUIReport */
```



Converter Tokens (7 of 10)

`/*V8:ConvertMode=Report */`

- ▲ **Should be used for inquiry and report programs that are not yet compatible** with the QAD Enterprise Applications FullGUIReport program template.
- ▲ This directive tells the converter that the program being converted is a report program that does not quite fit the standard model
- ▲ Programs converted as Report function the same as FullGUIReport but do not appear as "GUI" on the screen:
 - These reports do not have a print button
 - The output destination is in the same frame as the report parameters

Converter Tokens (8 of 10)

`/*V8:ConvertMode=ReportandMaintenance */`

- This converter token should *only be* used in sub-programs and include files which include UI functionality (frame definitions, update, set, prompt, display, etc.) where the source file is 'shared' by both Maintenance and Report type programs.
- It tells the GUI Converter that the program being converted to have both Report and Maintenance attributes.

Example: mfsotr1a.i

QAD Proprietary

BPC-08-PS-830

When:

- A Maintenance program: mymaint.p with a ConvertMode=Maintenance token and a report/inquiry program: myinquiry.p with a ConvertMode=Report both call the same sub-program or use the same include file, and
- That sub-program or include file has UI functionality

Then the sub-program or include file must use the ConvertMode=ReportAndMaintenance token.

Converter Tokens (9 of 10)

/*V8:ConvertMode=NoConvert */

- This token is used for files that the converter does not need to modify to be WINDOWS compatible
- Programs with no user interface or files that recognize the user interface without GUI converter help, such as browses

Example: mfglobe.I

Converter Tokens (10 of 10)

`/*V8:ConvertMode=ConditionalIncludeExclude */`

- ▲ When this ConvertMode token is found, the only processing done to produce the output file is to move mfdeclre/mfdtitle and process the `/*V8! ... */` and `/*V8-*/ ... /*V8+*/` directives

GUI Programming Considerations



GUI Programming Considerations

- ▲ A program must be either non-printing (maintenance/ data-entry) or printing (report) with a single input frame named "frame a"
- ▲ If a maintenance program has "Print Audit Trail Report" capability, put all the output commands (mfselprt.i too) into a separate subprogram and run it from the maintenance program (dmaprv.p and dmaprvrp.p demonstrate this)
- ▲ When aligning text vertically on a form use the AT, TO and COLON keywords; use COLON for labeled variables and AT for unlabeled variables
- ▲ Never place frames above line 3
- ▲ Put VALIDATE statements in the FORM, not in the data-entry statement

Reference: GUI Converter

- ▲ For more details on GUI Converter, refer to:

Appendix D: GUI Converter



Features of Desktop

- ▲ Full QAD Enterprise Applications access from a Web Browser
- ▲ Graphical HTML screens for almost all QAD Enterprise Applications menu items
- ▲ Intuitive navigation
- ▲ Menu search capability
- ▲ Graphical process diagram
- ▲ Customizable links and other customizable features
- ▲ Detachable screens
- ▲ Multiple output options, including browse output to a spreadsheet
- ▲ Extensive integrated help

Desktop 2 Development Standards



Desktop 2 Development Standards

- ▲ This section details the new development standards for PROGRESS 4GL code as a result of the Desktop 2 HTML interface for QAD Enterprise Applications eB and eB2
- ▲ Any creation or modification of Progress programs in these versions of QAD Enterprise Applications should follow the standards described in this section



Coding for HTML Screen

Two main points to consider when coding for Desktop HTML screen:

- ▲ HTML screen & Progress code interaction
 - The Submit and Go Back function, same as F1 and F4 in CHUI
 - The next/previous functions enabled by the up/down arrows
 - The right-click menu functions such as insert, delete and add

Coding for HTML Screen

- ▲ Navigation and performance
 - Having multiple set/updates in frames can become cumbersome and cause long delays in large programs.
 - For example: SO Maintenance requires a user to hit F1 at least 30 times to complete a simple 2 line sales order. In HTML this can cause long delays. So when creating CHUI screen, keep in mind that the same navigation will occur in HTML and that it is important to streamline it.



Use Lookup Created Using Browse Maintenance

- ▲ Lookups created using Browse Maintenance contain infrastructure needed by the Desktop HTML screens.
- ▲ The HTML screens cannot execute a scrolling window as the help key lookup



Use gpbranch.p to Handle Branched Lookups

- ▲ In QAD Enterprise Applications there are certain fields that need to run different lookups depending on certain conditions; that is, a field may need one lookup under condition A and another lookup under condition B
- ▲ One lookup will be hooked up to the field using Window Help Maintenance: this is the lookup for condition A. gpbranch.p is used in the maintenance code to assign the appropriate lookup based on condition
- ▲ gpbranch sets qad_wkfl record with the lookup information that is then used by the lookup launch mechanism to determine if the default lookup, which was attached using Window Help Maintenance, should be used or an alternate lookup should be run



Use gpbranch.p to Handle Branched Lookups

▲ Input parameters to gpbranch.p

- pFirstBrowse

The name of the default lookup for the field which is specified as Procedure to Execute in Window Help Maintenance

Mandatory and must be always supplied

- pSecondBrowse

The name of the alternate lookup to run for the field. If blank, default is run.

Note: gpbranch must be called before the field is enabled for update



Use gpbranch.p to Handle Branched Lookups

Example:

If source_entity_code = "1" then do:

```
{gprun.I ""gpbranch.p"" "(input `gplu360.p',  
input ` ')"}
```

end.

else do:

```
{gprun.I ""gpbranch.p"" "(input `gplu360.p',  
input `gplu348.p')")}
```

end.



Use gpbrparm.i to Pass Parameters to Lookups

- ▲ Used when Lookups created using Browse Maintenance need runtime data which is not stored in any of the global variables (like `global_part`, `global_site` etc) to provide proper filter for the query

QAD Proprietary

BPC-08-PS-970

Lookups created using Browse Maintenance (36.20.13) often need runtime data to provide proper filter for the query. Passing data from the Progress program to the Lookup is limited since the Lookup is not directly run from the Progress program but is run using a separate mechanism.

Often the Lookup needs to filter on a value in one of the QAD Enterprise Applications global variables such as `global_site` or `global_part`, and so on. This is fine and can be directly used in the Lookup. However, at other times, the Lookup will need data that is not in any of the global variables (for example, values of current screen fields or values retrieved from database fields). For these cases, `gpbrparm.i` is used.



Use gpbrparm.i to Pass Parameters to Lookups

- ▲ Parameters that can be set in maintenance programs using gpbrparm.I. These parameters can be used directly in the query of the lookup.

Character parameters	Decimal parameters	Date parameters	Logical parameter
c-brparm1	n-brparm1	d-brparm1	l-brparm1
c-brparm2	n-brparm2	d-brparm2	
c-brparm3	n-brparm3		
c-brparm4			
c-brparm5			

QAD Proprietary

BPC-08-PS-980

Any number of parameters can be set for a particular lookup by calling gpbrparm.i once for each parameter.



Use gpbrparm.i to Pass Parameters to Lookups

▲ Usage of gpbrparm.i

- {gpbrparm.i &browse=lookup &parm=parm &val=value}
- where,

lookup is the name of the lookup program

param is the name of one of the parameters listed in the previous slide

value is the value you want to assign to the parameter
– can be a field, a variable or a literal string

▲ Example:

```
/*Setting a parameter to a program variable */
{gpbrparm.i &browse=mglu066.p &parm=c-brparm1
  &val=mfguser}
```

```
/*Setting a parameter to a literal value */
{gpbrparm.i &browse=mglu066.p &parm=c-brparm1
  &val=""operator""}
```

Use gpbrparm.i to Pass Parameters to Lookups

- ▲ Limitations while using gpbrparm.i
 - Try to avoid placing it inside an editing block
 - Place it outside of any editing block and before the update/prompt-for/set statements of the fields that will be using the lookup



Use of Progress Browse Widgets

- ▲ Do not use Progress browse widget in maintenance programs
 - This does not refer to browse widgets in lookups or power browses
 - This only refers to the cases where browse widget is being used in maintenance frame

QAD Proprietary

BPC-08-PS-1010

A Progress engine walks the QAD Enterprise Applications screen widget tree collecting information about the widgets so that it can render them in HTML. The engine does not handle browse widgets. If a browse widget is used in a frame in QAD Enterprise Applications, it will not appear in the Desktop 2 HTML screen.

Labels and Fields

- ▲ Do not separate labels from fields
 - Progress allows to turn off a field label and replace it by adding character variable containing label to the frame. This should be avoided. This applies to both side labels and column labels.
 - The Desktop 2 HTML engine will not properly align or format label text that is not attached to a field



Labels and Fields

▲ Acceptable workaround on how to separate column labels from fields:

Define variable x as character no-undo.

Define variable y as character no-undo.

Define variable columnlabels1 as character no-undo.

Define variable columnlabel2 as character no-undo.

Assign columnlabel1 = "Field1"

columnlabel2 = "Field2".

Form

columnlabel1 columnlabel2 skip

x y

With frame a no-labels.

Display columnlabel1 columnlabel2 with frame a.

Update x y with frame a.

Labels and Fields

▲ Incorrect

Define variable x as character no-undo.

Define variable y as character no-undo.

Define variable columnlabel as character format

"x(24)" no-undo.

Columnlabel = "Field1 Field2".

Form

 columnlabel skip

 x y

With frame a no-labels.

Display columnlabel with frame a.

Update x y with frame a.

Field Enabling

- ▲ Do not use the Progress enable statement along with wait-for statement which allows fields to be enabled in more than one frame at a time
- ▲ Enable fields in only one frame at a time
- ▲ If more than one frame has enabled fields within it at a time, Desktop 2 HTML screens develop framing and focus issues

Field Enabling

▲ Incorrect:

Define variable x as character no-undo.

Define variable y as character no-undo.

enable x with frame a.

enable y with frame b.

Wait-for close of current-window.



Modifying Screens Within an Editing Block or Trigger

- ▲ Avoid making changes to the screen within as editing block
- ▲ These changes will not occur in Desktop HTML screen in the same manner as they appear in ChUI screen

QAD Proprietary

BPC-08-PS-1070

In CHUI, when a screen is changed in an editing block or trigger, the user sees the changes immediately. This includes label changes, screen value changes, fields appearing or disappearing, messages displaying, new frames appearing or any other screen modifications. In Desktop 2, these changes will not appear to the user or will appear at a later time, out of sequence.



Use gpwait.i to Pause for Display-Only Frames

- ▲ In QAD Enterprise Applications maintenance programs, use gpwait.i to explicitly pause the display of frames that do only data display and no prompting
- ▲ The Desktop engine that retrieves information from Progress character screen cannot execute properly when the system is paused. Progress triggers fire the engine code; however Progress will not execute the triggers when system is paused.
- ▲ gpwait.i prevents the pause statement from occurring by using a wait-for statement to simulate a pause, which returns control to the Desktop engine and allows triggers to be fired

QAD Proprietary

BPC-08-PS-1080

A pause can occur in several ways.

- Explicitly when a programmer adds a pause statement
- Automatically if frame is about to be hidden
- Automatically in down frames when the display has reached the end of the frame and is about to return to the top of the frame to display more data

In these cases, gpwait.i should be used to explicitly pause the display.



Use gpwait.i to Pause for Display-Only Frames

Example 1: DOWN FRAMES

```
for each cm_mstr no-lock:
    display cm_addr with frame a down.
    {gpwait.I &INSIDELOOP=YES
     &FRAME=a}
end.
{gpwait.I &OUTSIDELOOP=YES}
```

QAD Proprietary

BPC-08-PS-1090

For down frames that automatically pause after displaying a full screen of data and before displaying the next full screen, put one instance of the include before the end statement of the display loop and one after as shown in the example above.



Use gpwait.i to Pause for Display-Only Frames

Example 2: NON-DOWN frame about to be hidden

A frame that's been already displayed and about to be hidden will have an automatic pause that occurs before hiding. In this case, do as below:

Display cm_addr with frame a.
{gpwait.i}
Hide frame a.



Use gpwait.i to Pause for Display-Only Frames

Example 3: NON-DOWN FRAME with explicit pause

A frame that has been displayed and needs an explicit pause after it should be handled as below:

display cm_addr with frame a.

```
{gpwait.I &INCLUDEPAUSE=YES}
```

This will insert a wait-for when the Desktop is running and a pause statement when CHUI is running

Progress Selection-List Widget and the Desktop

- ▲ When an QAD Enterprise Applications program uses a Progress selection-list widget, that widget will not be displayed in the Desktop HTML version of the program
- ▲ Alternative is to use scrolling window include files like ecwindow.i, windowtxt.i, window.i, csscrad1.i etc
- ▲ Scrolling window includes can be used instead of Progress selection-list widget or can be used conditionally to run when the program is running in Desktop



Progress Selection-List Widget and the Desktop

- ▲ For example: Item Cost Maintenance uses csscrad1.I in program ppptmtd.p to provide the user with a list of items that can be scrolled through and selected



Defining Program Information for Custom Programs

- ▲ The records in the Program Information Table determine what type of interface a program will have in Desktop 2
- ▲ Browsers and lookups:
- ▲ A record should exist with "Web Logic Implemented = yes" and "Type=<blank>"
- ▲ All other programs (Maintenance,report,inquiry):
- ▲ "Web Logic Implemented = no"
- ▲ If "Type=<blank>" then program will run as a telnet screen
- ▲ If "Type=desktop" then program will run as an HTML screen.
- ▲ After modifying the program, add a record to Program Information Maintenance



Defining Program Information for Custom Programs

- ▲ When you change/add records in Program Information Maintenance, you must rebuild your menus and the search database; otherwise, your changes will not appear when you log-in to Desktop

```

mgpgmimt.p b+      36.3.21.1 Program
Information Maintenance 03/06/01
R,////////////////////T
.                  Program: popoxu
.
.      Selection Label: Purchase Order
Maintenance
F,////////////////////G
R,////////////////////T
.                  GIF File Name:
.
.      Web Logic Implemented: No
.
.                  Type: desktop
.
.      Presentation Text File:
F,////////////////////G

```

Using Search to Find Programs in Desktop

- ▲ Search in Desktop is based on information entered in Menu System Maintenance including menu number, menu label, short name and execution file
- ▲ You can use Search to:
 - Find programs based on one or more menu label keywords
 - Execute a specific program based on program name or menu number
 - Execute a specific program based on a shortcut
 - Find and display a specific menu in the menu system



Using Search to Find Programs in Desktop

- Result of entering sales in search and click on Go or press Enter key:

Number of matches displays here.

The document icon represents a program.

Folders represent menus.

Search Results: 1 - 25 Found (102) "sales"					Next
	Label	Program	Number	Name	Substitute
	Build Sales Analysis	utsabild.p	7.17.25.2		
	EC Sales Listing	vtvterp.p	2.13.15.8		
	Receipts - Sales Order Return	icsorc.p	3.10		
	Sales Account Browse	ppbr012.p	1.2.18		
	Sales Account Inquiry	pppsiq.p	1.2.10		✓
	Sales Account Maintenance	pppsmt.p	1.2.17		
	Sales Account Report	pplcrp.p	1.2.19		
	Sales Analysis Delete/Archive	sasaup.p	7.17.23		
	Sales Analysis Menu		7.17		
	Sales Analysis Utilities		7.17.25		
	Sales and Use Tax Interface Menu		36.5.3		
	Sales by Customer Browse	sasbr001.p	7.17.13		
	Sales by Customer Inquiry	sasbr003.p	7.17.13		✓
	Sales by Customer Report	sasbr003.p	7.17.14		
	Sales by Item Browse	sasbr007.p	7.17.17		
	Sales by Item Inquiry	sasbr004.p	7.17.17		✓
	Sales by Item Report	sasbr005.p	7.17.18		
	Sales by Salesperson Browse	sasbr002.p	7.17.5		✓



Using Search to Execute Custom Programs

- ▲ You can execute a program that is not in the menu but is in your PROPATH
- ▲ To do this, end the program name with an exclamation mark (!). This tells the system that this program is not on the menu
- ▲ To run custom.p from the search field, enter the following:

Custom.p!

Note: For a user to execute a program which is not in the menu, Program Execution must be set to Yes for that user in User Option Maintenance

QAD Proprietary

BPC-08-PS-1180

PROPATH is the environment variable containing the list of directories Progress searches for when looking for a program to execute.

Exercise: 4 Desktop 2



Exercise 4, Desktop 2 (20 minutes)

- ▲ Update your program of Exercise 2 (Functionality Reuse for Desktop 2)
- ▲ Define your program as a Desktop 2 program using Program Information Maintenance (or even better via Program Information Update)
- ▲ Test it via Desktop 2 (if available)
 - Save <myProg.p>
 - Create language and two-letter directory in your working directory (mkdir us us/<my>)
 - Compile <myProg.p> save into us/<my>
 - Make sure that you can start non-defined programs in Desktop 2 (User Option Maintenance, Program Execution should be yes)
 - Start IE 5.5 and select your Desktop 2 environment
 - Start program by typing <myProg>! in the menubar of Desktop 2

Desktop 2 and pgmi Records

- ▲ Every QAD Enterprise Applications menu item needs to have an associated pgmi record
- ▲ If you add a menu item to eB2, use 36.7.1 R&D Program Information Maintenance to create pgmi_mstr record for it with type field set to "desktop" and Web Logic Implemented = NO. This will ensure that your program will run as a Desktop 2 HTML screen.

Translations



QAD Enterprise Applications Translations

- ▲ QAD Enterprise Applications Translation Programming Standards are covered in:

Chapter 16 – Multi-Language Labels

Performance Considerations



Performance Considerations

- ▲ EXCLUSIVE-LOCKS on quick deletes
 - Reduces the number of times the record is read
- ▲ Do:

```
FOR EACH qad_wkfl WHERE qad_key1 = "mrp/drp"
EXCLUSIVE-LOCK:
    DELETE qad_wkfl.
END.
```
- ▲ Don't:

```
FOR EACH qad_wkfl WHERE qad_key1 = "mrp/drp":
    DELETE qad_wkfl.
END.
```

Performance Considerations

- ▲ The NOT operator in a WHERE clause
 - Can cause full table scans
 - Avoid using “WHERE NOT logical_var”
 - Use “WHERE logical_var = FALSE”

Performance Considerations

▲ LOOK AHEAD QUERIES

- Limits the number of select statements generated for a particular 4GL database access statement
- FOR [EACH|FIRST|LAST] record-phrase
- DEFINE QUERY/OPEN QUERY
- FIND statements do not use look ahead queries



Performance Considerations

- ▲ Replace FIND FIRST/FIND NEXT
 - Generates two SQL select statements
 - select progress_recid where...
 - select <all fields> where progress_recid =...
 - A better option would be a QUERY statement

Performance Considerations

▲ Replace FIND FIRST/FIND NEXT

▲ Don't:

```
FIND FIRST pt_mstr NO-LOCK WHERE pt_part >= "123" NO-  
ERROR.  
REPEAT WHILE AVAILABLE PT_MSTR:  
FIND NEXT pt_mstr NO-LOCK WHERE pt_part >= "123" NO-  
ERROR.  
END.
```

▲ Do:

```
DEFINE QUERY q_pt_mstr FOR pt_mstr.  
OPEN QUERY q_pt_mstr FOR EACH pt_mstr NO-LOCK  
WHERE pt_part >= "123".  
GET FIRST q_pt_mstr.  
REPEAT WHILE AVAILABLE PT_MSTR:  
GET NEXT q_pt_mstr.  
END.
```



Performance Considerations

▲ Replace FIND FIRST/FIND LAST

▲ Don't:

```
FIND FIRST pt_mstr NO-LOCK  
WHERE pt_part >= "123" NO-ERROR.  
IF AVAILABLE pt_mstr THEN ...
```

▲ Do:

```
FOR FIRST pt_mstr NO-LOCK WHERE pt_part  
  >= "123":  
  END.  
IF AVAILABLE PT_MSTR THEN ...
```

Performance Considerations

- ▲ Use Field Lists to reduce the size of the query
 - Best used on NO-LOCK queries
 - Ignored on EXCLUSIVE-LOCK queries

- ▲ Don't:

```
FOR FIRST pt_mstr NO-LOCK WHERE pt_part >= "123":  
  END.  
IF AVAILABLE PT_MSTR THEN ...
```

- ▲ Do:

```
FOR FIRST pt_mstr FIELDS(pt_part pt_desc1) NO-  
  LOCK  
WHERE pt_part >= "123": END.  
IF AVAILABLE PT_MSTR THEN ...
```



Performance Considerations

▲ Field Lists

- Can use the xref compile option to ensure a complete field list
- **WARNING:** missing fields in a field list cause RUNTIME ERRORS

Performance Considerations

- ▲ How to determine field lists?

In the PROGRESS editor, run the following command.

Compile *<filename>* xref *xreffilename*.

At the unix prompt, run the following script to get the list of fields:

```
grep "ACCESS pt_mstr" xreffilename | while read a
do
    b=`echo $a | awk '{print $6}'`
    echo $b
done | sort -u
```



Performance Considerations

- ▲ **SHARED BUFFERS or BUFFERS passed as PARAMETERS**
 - Allow records to be read once
 - Replaces the shared recid variables
 - Easily implemented by using a buffer with the same name as your table name

Performance Considerations

- ▲ Replace old shared RECID variables
 - Each program does a select on the same record
 - Remember our goal to reduce DB access?

```
/*pgm1.p*/  
DEFINE NEW SHARED VARIABLE pt_recno AS RECID.  
FOR FIRST pt_mstr NO-LOCK WHERE pt_part = "123": END.  
pt_recno = RECID(pt_mstr).  
{gprun.i ""pgm2.p""}
```

```
/*pgm2.p*/  
DEFINE SHARED VARIABLE pt_recno AS RECID.  
FOR FIRST pt_mstr NO-LOCK  
  WHERE RECID(pt_mstr) = pt_recno: END.  
DISPLAY pt_part.
```



Performance Considerations

▲ Shared Buffers

- This is better
 - We get rid of the extra select

```
/*pgm1.p*/  
DEFINE NEW SHARED BUFFER pt_mstr FOR pt_mstr.  
FOR FIRST pt_mstr NO-LOCK WHERE pt_part = "123":  
    END.  
{gprun.i ""pgm2.p""}  
  
/*pgm2.p*/  
DEFINE SHARED BUFFER pt_mstr FOR pt_mstr.  
DISPLAY pt_part.
```


Performance Considerations

▲ Passing a Buffer as a Parameter

- EVEN BETTER!
 - We get rid of the shared variables!

```
/* pgm1.p */  
DEFINE BUFFER pt_mstr FOR pt_mstr.  
FOR EACH pt_mstr WHERE pt_part >= "1" AND pt_part <=  
  "2":  
  RUN pgm2.p (BUFFER pt_mstr).  
END.
```

```
/* pgm2.p */  
DEFINE PARAMETER BUFFER pt_mstr FOR pt_mstr.  
DISPLAY pt_part.
```



Performance Considerations

▲ Internal Procedures

- Breaks up a program's r-code segments
- Best used in place of include files or added external procedures/sub-programs

```
RUN my_procedure.  
.   
.   
PROCEDURE my_procedure:  
  FOR EACH pt_mstr FIELDS(pt_part pt_desc1  
    pt_desc2)  
    NO_LOCK WHERE pt_part >= "123":  
      DISPLAY pt_part pt_desc1 pt_desc2.  
  END.  
END. /* my_procedure */
```

Performance Considerations

▲ ASSIGN statements

- Group field/variable assignments into one statement
- Reduces the r-code segment size

▲ Don't:

```
so_cust = cust.  
so_bill = cust.  
so_ord_date = TODAY.
```

▲ Do:

```
ASSIGN so_cust = cust  
      so_bill      = cust  
      so_ord_date = TODAY.
```



Performance Considerations

- ▲ NO-UNDO variable definitions
 - More efficient
 - Should be the default method of variable definitions
 - If you need undo-processing, document how it is used in the program comments

Performance Considerations

- ▲ NO-UNDO variable definitions
 - With NO-UNDO my_var = 10 when you press ENDKEY at the update statement

```
/* my_var remains 10 when user presses the ENDKEY */  
DEFINE VARIABLE my_var AS INTEGER NO-UNDO.  
ASSIGN my_var = 0.  
  
DO TRANSACTION ON ENDKEY UNDO, LEAVE.  
    my_var = 10.  
    UPDATE my_var WITH FRAME a.  
END.  
  
DISPLAY my_var WITH FRAME b.
```



Performance Considerations

- ▲ NO-UNDO variable definitions
 - Without NO-UNDO my_var = 0 when you press ENDKEY at the update statement

```
/* Rollback my_var to zero when user presses endkey */  
DEFINE VARIABLE my_var AS INTEGER.  
ASSIGN my_var = 0.  
  
DO TRANSACTION ON ENDKEY UNDO, LEAVE.  
    my_var = 10.  
    UPDATE my_var WITH FRAME a.  
END.  
  
DISPLAY my_var WITH FRAME b.
```

QAD Proprietary

BPC-08-PS-1390

Defining a variable as NO-UNDO creates a savings in IO processing and thus better performance since the variable's data-writes to the local BI file are unnecessary.

Performance Considerations (1 of 2)

▲ Avoid Using Shared Variables

- Never introduce a shared variable in a new program/procedure. If another program or a procedure needs a variable, pass it as an input or input-output parameter
- The parameter could be of type input, input-output or output as demanded by the procedure design



Performance Considerations (2 of 2)

▲ Exceptions

- System-level shared variables like user-id, password etc and application-level shared variables like part, site, entity etc are exceptions to the standard
- Split-reports and inquiry programs at the top-level. It is okay to define selection-criteria variables at the first level split-programs as shared variables.
- Programs where shared variables are already used extensively across multiple programs

Performance Considerations

▲ Shared Variables example

sopro1:

```
define buffer bso_mstr for so_mstr.
define variable soCustomer as character no-undo.
..... important logic
{gprun.i `sopro2.p'
    "(bso_mstr,      /* buffer is always input-output */
    input soCustomer)"
}
.....more logic
```

Shared variables are hard to maintain and test when they are used in multiple programs and procedures. It violates the modular programming concept of coupling. It is a good design when there is least coupling between procedures/programs. Due to proliferation of shared variables, there were several cases of bugs not being detected until after product release.

Performance Considerations

soprogram2:

```
define parameter buffer pso_mstr for so_mstr.  
/*parameter is input-output*/  
define input parameter pSOCustomer as character  
no-undo.
```

```
.....some logic  
for first cm_mstr where cm_addr = pSOCustomer  
  no-lock:  
end.  
if available cm_mstr then do:  
  .....more logic
```

Performance Considerations

- ▲ Variables used to prompt for options
 - Logical data type fields can be used when there can be only two choices and those options are direct opposites of each other. For example, the field's Progress format may be "yes/no", "on/off", "high/low", "detail/summary", etc.
- ▲ Integer data type fields do provide more than two options to the user, but are generally not preferred because they do not adequately represent the true value of the option. In many cases, it is not always clear what 1 or 2 means when presented as an option.



Performance Considerations (1 of 2)

- ▲ Character data type fields are the most flexible in terms of their use for options
- ▲ They allow for more than two options and do not require schema changes to add options in the future
- ▲ Character field options do not need to be opposites of each other since next/previous logic and lookups can be attached to their user interface to aid the user in making a choice

Performance Considerations (2 of 2)

- ▲ To allow the options to be presented in a language appropriate to the user, the designer can choose one of the following methods:
 - Language details provide translated values for a fixed number of options. The options are determined by the designer and the values are initially provided by QAD. Underlying numeric codes allow for business logic to be created for specific options. The QAD Enterprise Applications customer can then change the option values after the data is installed.
 - Generalized codes provide unlimited number of options in the user's own language. The options are determined by the QAD Enterprise Applications customer, not the QAD designer and hence usually do not carry any specific logic per option.



Performance Considerations

Variable `addressType` defined as character with language details:

```
Numeric Code = "1"; Mnemonic = "Customer";  
Label = "Customer Address Type"  
Numeric Code = "2"; Mnemonic = "Supplier";  
Label = "Supplier Address Type"
```

Variable `billingLevel` defined as logical with a format of "Detail/Summary"

Performance Considerations

- ▲ Remove FIND statements that are no longer needed
 - Many FIND statements may have become useless after a patch has commented out the record's use



Performance Considerations

▲ Reconsider your queries

- Queries that are slow on PROGRESS are SLOWER on Oracle
- Perhaps there is a better way to find your data

```
/*old*/  
FOR EACH si_mstr WHERE si_site = "101" NO-LOCK:  
    FOR EACH pt_mstr WHERE pt_part = "bolt" NO-  
    LOCK:  
  
/*new*/  
FOR EACH in_mstr WHERE in_site = "101" AND  
    in_part = "bolt" NO-LOCK:
```


Exercise: Custom Programming

Add the new fields, C1AD_EMAIL and C1AD_EMAIL2, to the Customer Maintenance (2.1.1) data entry screen

- Where does it logically fit?
- What program gets modified?
- Make the modifications

Solution: Custom Programming

- ▲ Current Customer Maintenance Screen
 - The e-mail address information really belongs in frame A

Frame A

```

xterm
adcsmt.p e+ 2.1.1 Customer Maintenance 10/07/98
Customer Address
Customer: 0001000
*Name: Reliable Central Purchasing
Address: Attention: Accounts Payable
Address: 300000020 Century Blvd
Address: P.O. Box 675
City: Los Angeles State: CA Post: 89000 Format: 0
Country: United States of America County: 001
Attention: Mark Schnell [2]: Max Nichts
Telephone: 213-920-0394 Ext: [2]: Ext:
*Fax/Telex: 213-920-0399 [2]: Added: 05/15/87

Customer Data
Sort Name: Reliable Central Purchasing Type: 1 Taxable: no 1
Salespsn1: JJP Multiple: no Region: US Price Tbl:
Ship Via: Currency: USD Disc Tbl: 10
AR Acct: 1200 Language: Fixed Price: yes
Resale: 89-02-03 X82D Site: 10000 Class:
Remarks: BILLING ONLY - USE SHIP TO ON ALL ORDERS Partial OK: yes

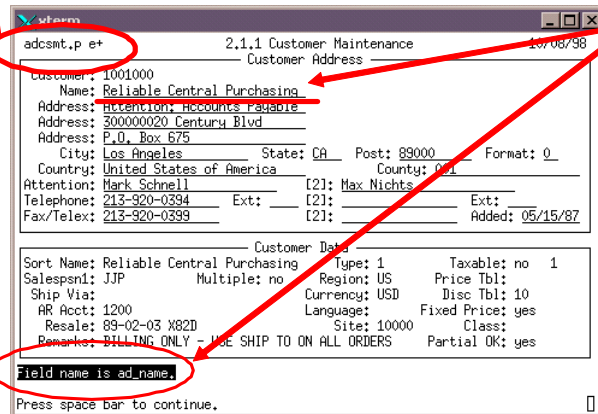
F1=Go 2=Help 3=Ins 4=End 6=Menu 7=Recall 8=Clear 9=Prev 10=Next
  
```

Move the "Name" field up on the same line as the "Customer" field. This will move all the lines up... and maybe we could fit the e-mail addresses on the last line, underneath the "Fax/Telex" fields.

Solution: Custom Programming

▲ Where to start?

- Start at the main program and go from there
- The program's name appears at the top left hand corner of the screen



adcsmt.p e+ 2.1.1 Customer Maintenance 18/06/98

Customer: 1001000
 Name: Reliable Central Purchasing
 Address: Atencion Accounts Payable
 Address: 300000020 Century Blvd
 Address: P.O. Box 675
 City: Los Angeles State: CA Post: 99000 Format: 0
 Country: United States of America County: 001
 Attention: Hark Schnell [2]: Max Nichts
 Telephone: 213-920-0394 Ext: [2]: Ext:
 Fax/Telex: 213-920-0399 [2]: Added: 05/15/87

Customer Data:
 Sort Name: Reliable Central Purchasing Type: 1 Taxable: no 1
 Salespsn1: JJP Multiple: no Region: US Price Tbl:
 Ship Via: Currency: USD Disc Tbl: 10
 AR Acct: 1200 Language: Fixed Price: yes
 Resale: 89-02-03 X82D Site: 10000 Class:
 Remarks: BILLING ONLY - USE SHIP TO ON ALL ORDERS Partial OK: yes

Field name is ad_name.
 Press space bar to continue.

With the cursor on the "Name" field, pressing Ctrl-F tells me that I am editing field AD_NAME.

Programming Lab



Solution: Custom Programming

- ▲ Indicating the customization revision level
 - Keep track of revisions for each program modified and rename programs as you customize

```
/*custom adcsmt.p*/
```

```
/* cladcsmt.p - custom CUSTOMER MAINTENANCE          adcsmt.p*/
/* My Company Inc.                                   */
```

```
/* adcsmt.p - CUSTOMER MAINTENANCE                    */
.
```

```
.
```

```
.
```

```
/* REVISION: 8.5      LAST MODIFIED: 09/05/96   BY: APP *G2DL**/
```

```
/* REVISION: 8.5      LAST MODIFIED: 10/07/96   BY: SXN *H0N3**/
```

```
/* REVISION: 8.5      LAST MODIFIED: 07/17/97   BY: SXN *J1VT**/
```

```
/* CUSTIMIZATION: 1.0 LAST MODIFIED: 10/08/98   BY: AME *C001**/
```

```
/* DISPLAY TITLE */
```

```
{mfdtitle.i "e+*c1"} /*C001*/
```

```
define variable del-yn like mfc_logical.
```

```
define variable cmnbr like cm_addr.
```

Programming Lab Solution



Solution: Custom Programming

- ▲ Find where the frame is defined within this program
 - The form definition is generally located after the variable definitions. If it is not in the current program being edited, look for an include file that might contain the definition

```
.  
.  
/*G282*/ {gpaud.i &uniq_num1 = 01 &uniq_num2 = 02  
          &db_field = cm_mstr &db_field = cm_addr}  
/*G282*/ {gpaud.i &uniq_num1 = 03 &uniq_num2 = 04  
          &db_field = ad_mstr &db_field = ad_addr}  
  
          /* DISPLAY SELECTION FORM */  
/*H086*/ {adcsmt02.i}
```



Solution: Custom Programming

- ▲ Re-name programs to indicate they have been customized
 - adcsmt02.i ---> c1adcsmt02.i
 - adcsmt.p ---> c1adcsmt.p
- ▲ Reflect the new name in the calling program

/* c1adcsmt.p */

```
.
/*G282*/ {gpaud.i &uniq_num1 = 01 &uniq_num2 = 02
          &db_field = cm_mstr &db_field = cm_addr}
/*G282*/ {gpaud.i &uniq_num1 = 03 &uniq_num2 = 04
          &db_field = ad_mstr &db_field = ad_addr}

          /* DISPLAY SELECTION FORM */
/*C001* /*H086*/ {adcsmt02.i} */
/*C001*/ {c1adcsmt02.i}
```



Solution: Custom Programming

▲ Editing the FORM statement for FRAME A

```
/* c1adcsmt02.i */
```

```
FORM
```

```
SKIP
```

```
cm addr COLON 10
```

```
ad_name COLON 10
```

```
ad_line1 COLON 10
```

```
.
```

```
.
```

```
ad_fax COLON 10
```

```
ad_fax2 COLON 43 label "[2]"
```

```
ad_date
```

```
WITH FRAME a TITLE COLOR NORMAL " Customer Address "
```

```
SIDE-LABELS WIDTH 80 ATTR-SPACE.
```

```
/*C001* ad_name COLON 10 */
```

```
/*C001*/ ad_name COLON 43
```

```
/*C001*/ clad_email COLON 10 VIEW-AS FILL-IN SIZE 27 BY 1
```

```
/*C001*/ clad_email2 COLON 43 LABEL "[2]" VIEW-AS FILL-IN SIZE 27 BY 1
```



Solution: Custom Programming

▲ C1adcsmt.p

- Modify display statements during scrolling

/* c1adcsmt.p */

```
prompt-for cm_mstr.cm_addr editing:

/* FIND NEXT/PREVIOUS RECORD */
{mfnp.i cm_mstr cm_addr cm_addr cm_addr cm_addr cm_addr}

if recno <> ? then do:
    find ad mstr where ad addr = cm addr no-lock no-error.
/*C001*/ find clad mstr where clad addr = ad addr no-error.
display cm_addr ad_name ad_line1 ad_line2 ad_line3
/*F698* ad_city ad_state ad_zip ad_country */
/*F698* ad_city ad_state ad_zip ad_format ad_country
/*H075* ad_ctry when ({txnew.i}) */
/*H075* ad_ctry when ({txnew.i} or ec_ok )
ad_attn ad_phone ad_ext ad_attn2 ad_phone2 ad_ext2
ad_fax ad_fax2 ad_date
ad county
/*C001*/ clad_email clad_email2.
```

Find the
record,
update
the
display.

QAD

BPC-08-PS-1590



Solution: Custom Programming

▲ C1adcsmt.p

- Modify the Add, Modify Section

```
/*GJ30*/ do transaction with frame a on endkey undo, leave loopa:
/* ADD/MOD/DELETE */
find ad_mstr where ad_addr = cmnbr no-error.
if not available ad_mstr then do:
    create ad_mstr.
    assign ad_addr = cmnbr.
    ad_type = "customer".
    ad_date = today.
/*F698*/ ad_format = adc_format.
    new_cust = true.
/*GM11*/ recno = recid(ad_mstr).
end.
```

```
/*C001*/ find clad_mstr where clad_addr = ad_addr no-error.
/*C001*/ if not available clad_mstr then do:
/*C001*/     create clad_mstr.
/*C001*/     assign clad_addr = ad_addr.
/*C001*/ end.
```

If the
record
doesn't
exist,
create it.



Solution: Custom Programming

▲ C1adcsmt.p

- Modify the Add, Modify Section

/* c1adcsmt.p */

```
set1:
  do on error undo, retry:
    set ad_name ad_line1 ad_line2 ad_line3
/*H0N3*/ ad_city ad_state ad_zip ad_format ad_country
/*H0N3*/ when ((not {txnew.i}) and (not ec_ok))
/*H075*/ ad_ctry when ({txnew.i} or ec_ok) ad_county
          ad_attn ad_phone ad_ext ad_fax
/*C001*/ c1ad_email
          ad_attn2 ad_phone2 ad_ext2 ad_fax2
/*C001*/ c1ad_email2
          ad_date go-on (F5 CTRL-D).
```

Add the new fields to the update statement.

Put the fields in the order that you want them to be entered.

Here, we have the first e-mail entry with the first contact information, and the second e-mail entry with the second contact.



Solution: Custom Programming

- ▲ C1adcsmt.p
- ▲ Modify the Delete Section

/* c1adcsmt.p */

```
/*C001*/ find clad_mstr where clad_addr = cm_addr  
/*C001*/ exclusive-lock no-error.  
/*C001*/ if available clad_mstr then delete clad_mstr.
```

```
/* DELETE CUSTOMER MASTER */  
delete cm_mstr.  
clear frame a.  
clear frame b.  
del-yn = no.
```

Delete the
C1AD_MSTR
records along
with all the
other related
records



Solution: Custom Programming

▲ What other programs use adcsmt02.i?

- Cross Reference Menus
- Grep

```
$ cd $MFGPRO/us/src
$ grep adcsmt02.i *
adcsmt.p:/*H086*/ {adcsmt02.i}
adcsmt02.i:/* adcsmt02.i - CUSTOMER MAINTENANCE
FORMS..
adcsmt.p:          {adcsmt02.i}
```

- Edit one more program to reflect new name

```
.
.
define shared variable undo_adcsmtc like mfc_logical.
define shared variable errfree      like mfc_logical.

/* DISPLAY SELECTION FORM */
/*C001* {adcsmt02.i} */
/*C001*/ {cladcsmt02.i}
```

QAD Proprietary

BPC-08-PS-1630

Solution: Custom Programming

▲ C1adcsmt.p

- Update the call to adcsmt.p
 - adcsmt.p ---> c1adcsmt.p

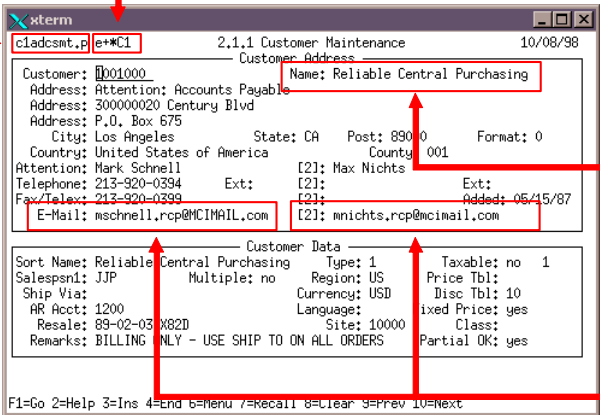
```

/* c1adcsmt.p */
/*H0MC*/ loopc:
/*H0MC*/ do on error undo, leave:
/*C001* /*H086*/      {gprun.i ""adcsmt.p"" }*/
/*C001*/      {gprun.i ""c1adcsmt.p"" }
/*H086*/      if not errfree then do:
/*H0MC** /*H086*/      undo innerloop, leave. */
/*H0MC*/      undo loopc, leave innerloop.
/*H086*/      end.
/*H0MC*/ end.

```

Solution: Custom Programming

- ▲ Custom Customer Maintenance Screen...
 - Try to keep the screen as “standard” as possible



The screenshot shows the '2.1.1 Customer Maintenance' screen in a terminal window. The screen displays customer information for 'Reliable Central Purchasing'. Annotations with red arrows point to specific fields and boxes:

- New custom program name:** Points to the command line 'c1adcsmt.p e*#C1'.
- Indicate the revision level within mfdtitle.i:** Points to the 'Name' field 'Reliable Central Purchasing'.
- Move the “Name” field up to the first line:** Points to the 'Name' field 'Reliable Central Purchasing'.
- Add the “e-mail” fields to the bottom of frame a:** Points to the 'E-Mail' field 'mschnell.rcp@MCMAIL.com'.

The screen content includes:

```

Customer: 0001000 Name: Reliable Central Purchasing
Address: Attention: Accounts Payable
Address: 300000020 Century Blvd
Address: P.O. Box 675
City: Los Angeles State: CA Post: 89000 Format: 0
Country: United States of America County: 001
Attention: Mark Schnell (2): Max Nichts Ext:
Telephone: 213-920-0394 (2): Added: 05/15/87
Fax: 213-920-0399 (2):
E-Mail: mschnell.rcp@MCMAIL.com (2): mnichts.rcp@mcimail.com

Sort Name: Reliable Central Purchasing Type: 1 Taxable: no 1
Salespnl: JJP Multiple: no Region: US Price Tbl:
Ship Via: Currency: USD Disc Tbl: 10
AR Acct: 1200 Language: Site: 10000 ixed Price: yes
Resale: 89-02-03 X82D Class:
Remarks: BILLING ONLY - USE SHIP TO ON ALL ORDERS Partial OK: yes
  
```

At the bottom, a command line is visible: F1=Go 2=Help 3=Ins 4=End 5=Menu 7=Recall 8=Clear 9=Prev 10=Next

Solution: Custom Programming

▲ What else?

- Data conversion/Creating records...
 - To support a AD_MSTR, C1AD_MSTR join (for custom reports, etc., you need to create a C1AD_MSTR record for every AD_MSTR record

```
FOR EACH ad_mstr NO-LOCK:  
    CREATE clad_mstr.  
    clad_addr = ad_addr.
```

END.

- Location of patches
 - Create a separate r-code/source code environment for customizations
 - \$MFGPRO/patch
 - \$MFGPRO/patch/us
 - \$MFGPRO/patch/us/src

Summary



Programming Standards

- ✓ Programming Naming Conventions
- ✓ Patch Marking Conventions
- ✓ Templates for Inquiries, Reports and Maintenance Programs
- ✓ Net UI Infrastructure
- ✓ QAD Enterprise Applications Programming considerations (General and Desktop)
- ✓ QAD Enterprise Applications Translations
- ✓ Performance Considerations

Chapter 9

Multi-Database Programming

Introduction



Multi-Database Programming

In this section, you learn the:

- ▲ Use of the Alias statement
- ▲ Use of Database functions
- ▲ Multi-database code samples

Alias Statement



Alias Statement

- ▲ Synonym for logical name
- ▲ Logical-name database must be connected

```
CREATE ALIAS alias  
FOR DATABASE logical-name  
[NO-ERROR] .
```

QAD Proprietary

BPC-09-MDP-024

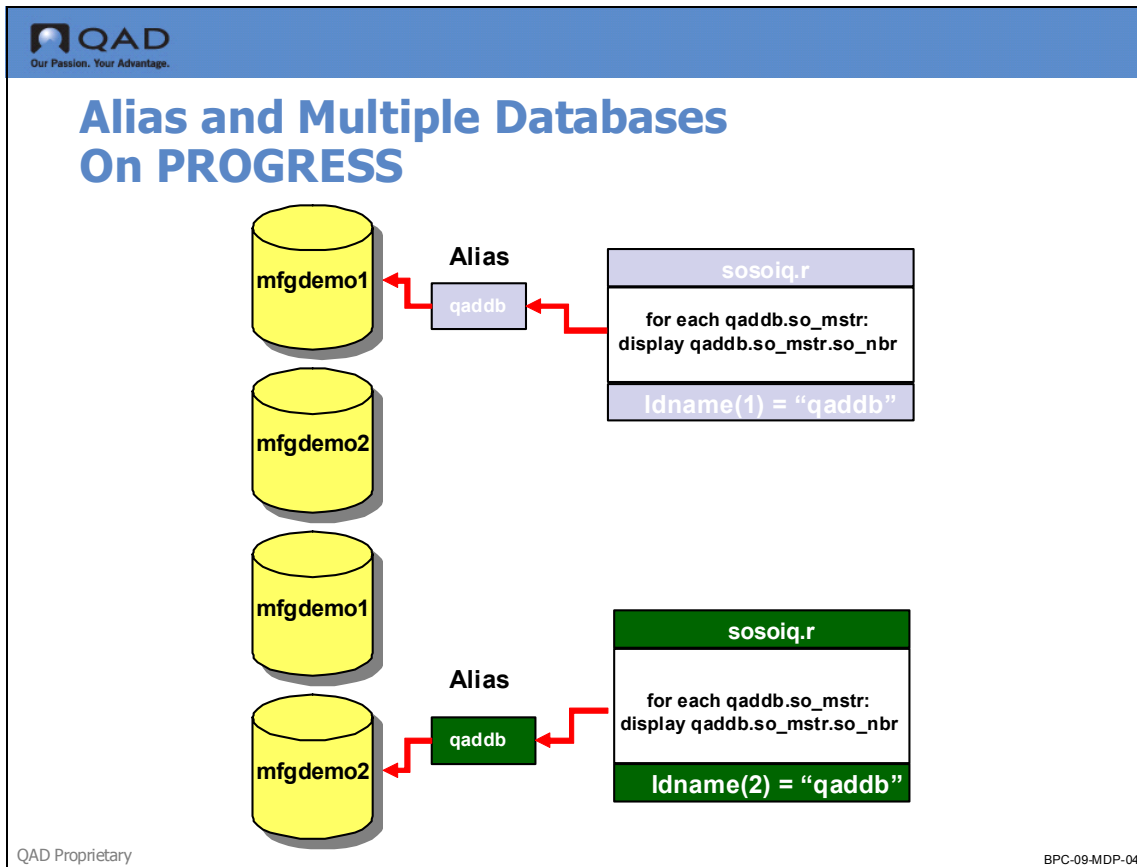
Alias is an unquoted string, quoted string, or character expression that represents an alias for a database.

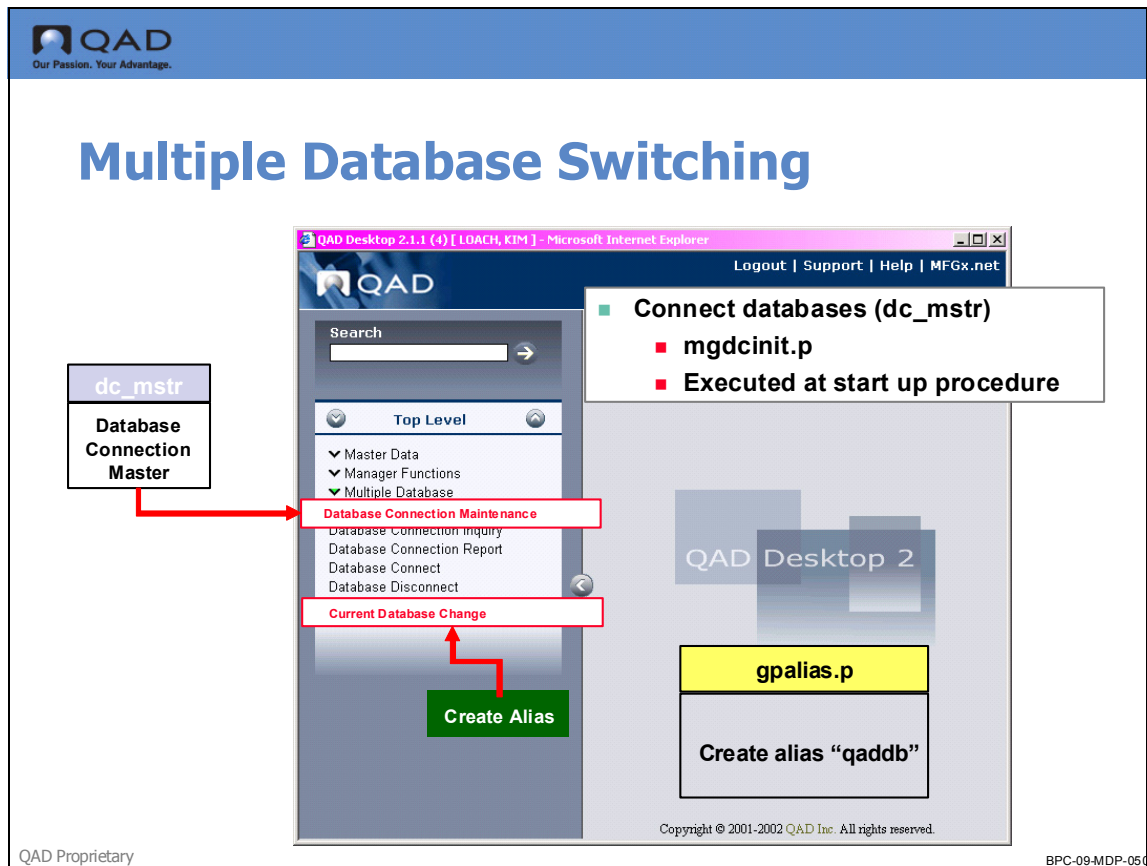


Use of Alias Statement

- ▲ Main purpose is to allow for a general purpose application to expect a specific database name
 - A database can have more than one alias, but each alias refers to only one database
 - Cannot assign and reference an alias within same procedure
- ▲ Allows an application to seamlessly switch to different databases within the same program

Alias and Multiple Databases





This does not refer to the three or four databases that are connected with each session login (mfg, adm, help, and so on); it refers to alternate 'mfg' database.

QAD
Our Passion. Your Advantage.

Database Connection Maintenance

The screenshot shows the 'Database Connection Maintenance' window. The 'Database Name' field is set to 'qaddb'. The 'Physical Database Name' field is also set to 'qaddb'. The 'Type' field is set to 'Progress'. The 'Active' checkbox is checked. The 'Host', 'Server', 'Network', and 'Parameter File' fields are empty. The 'Database Directory' field is also empty. The 'Delete', 'Back', and 'Next' buttons are at the bottom right.

Exec \$DLC/bin/_progres \
-db qaddb -ld qaddb...

The logical name in the startup script **MUST** match the "database name" in the database connection master (dc_mstr).
No matter what database platform, the "Type" must be **PROGRESS**.

Database Connection Maintenance (36.6.1)

QAD Proprietary BPC-09-MDP-060

Site Maintenance



Our Passion. Your Advantage.

Site Maintenance

Site Maintenance X

Go To Action

Site: 10000

Exec \$DLC/bin/_progres \
-db mfgdemo -ld **mfgdemo** ...

Description: San Diego Main Plant

Entity: T100

Default Inventory Status:

Automatic Locations: ☒

Domain: mfgdemo

EMT Supplier:

External Supplier:

Transfer Variance Acct: 5030

Transfer Ownership: ☐

The logical name in the startup script MUST match the “database name” in the database connection master (dc_mstr).

The database referenced in the Site Master (si_mstr) MUST match the database referenced in both the dc_mstr and the startup script.

Delete

Back

Next

Site Maintenance (1.1.13)

QAD Proprietary

BPC-09-MDP-07

Progress 4GL Functions to Use

▲ LDBNAME

- Returns the logical name of a currently connected database

`LDBNAME (integer-expression | logical-name | alias)`

▲ PDBNAME

- Returns the physical name of a currently connected database

`PDBNAME (integer-expression | logical-name | alias)`

▲ DBTYPE

- Returns the database type of a currently connected database (PROGRESS, RMS, ORACLE, etc.). DBTYPE accepts an integer expression or a character expression as a parameter

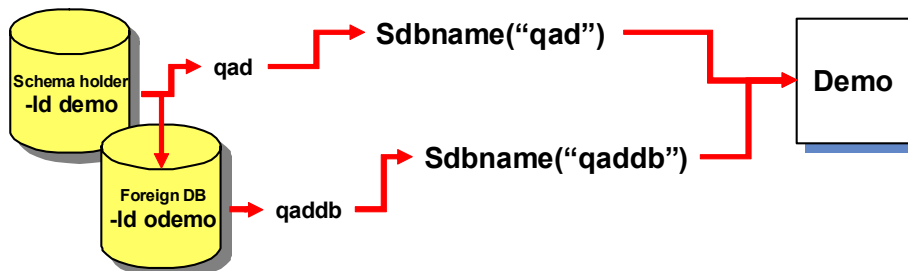
`DBTYPE (integer-expression | logical-name | alias)`

Progress 4GL Functions to Use

▲ SDBNAME

- Accepts an integer expression or a character expression as a parameter. If the parameter resolves to a currently connected non-PROGRESS database then SDBNAME function returns the logical name of the schema holder database containing the non-PROGRESS schema. If the parameter resolves to a currently connected PROGRESS database, the SDBNAME function returns the logical name of this database.

`SDBNAME (integer-expression | logical-name | alias)`



Multi-Database Code Samples

▲ mf.p (partial listing)

For PROGRESS
databases, you
just need a
simple create
alias statement

For Foreign
databases, you
need to set the
aliases in pairs.
Use the
SDBNAME
function to help

```

if num-dbs > 0 then do:
  if pdbname("qaddb") = ? then do:
    if not can-do(gateways,"oracle") then
      create alias qaddb for database
      value(ldbname(1)).
    else do:
      do sdb = 1 to num-dbs:
        if dbtype(sdb) <> "PROGRESS" and
          sdbname(sdb) = sdbname(ldbname(1)) then
          do:
            create alias qaddb for database
            value(ldbname(sdb)).
            create alias qad for database
            value(ldbname(1)).
            leave.
        end.
      end.
    if ldbname(1) <> sdbname("qaddb") then
      create alias qaddb for database
      value(ldbname(1)).
  end.
end....
  
```



Multi-Database Code Samples

▲ gpalias.p (partial listing)

For PROGRESS databases, you just need a simple create alias statement

For Foreign databases, you need to set the aliases in pairs. Use the SDBNAME function to help

```
find si_mstr where si_site = new_site no-lock no-error.
  if available si_mstr and si_db <> global_db then do:
    if NOT can-do(gateways,"oracle") then
      create alias qaddb for database value(si_db).
    else do:
      /* Assign qaddb to foreign DB and qad to the SH */
      do sdb = 1 to num-dbs:
        if dbtype(sdb) <> "PROGRESS" and
          sdbname(sdb) = sdbname(si_db) then do:
          create alias qaddb for database value(ldbname(sdb)).
          create alias qad for database value(si_db).
          end.
        end.
      /* if qaddb wasn't assigned to the si_db then assign it*/
      if si_db <> sdbname("qaddb") then
        create alias qaddb for database value(si_db).
      end.
    end.
  end.
```

galias.p



More on galias.p

- ▲ Switch alias by site
 - galias.p
 - Shared variable new_site
 - galias2.p - (galias.p with error checking)
 - Input parameter new_site
 - Output parameter error-flag
- ▲ Switch alias by database name
 - galias.i
 - Parameter ldbname
 - galiasd.p
 - Shared variable new_db
 - galias3.p - (galiasd.p with error checking)
 - Input parameter new_db
 - Output parameter error-flag

Using gpalias: An Example

```

/* multi01.p - Test multi-database */
{mfdtitle.i "test"}
define new shared variable new_site like si_site.
define variable old_site like si_site.
define variable qty_oh as decimal no-undo.
update new_site with frame a.
for each pt_mstr where pt_site = new_site no-lock:
  /* move alias qaddb to new_sites database */
  {gprun.i "gpalias.p"}

  /* get the quantity on hand for new_site, pt_part */
  {gprun.i "multi02.p"
    "(input pt_part, input pt_site, output qty_oh)"}

  display pt_part pt_desc1 pt_site global_db qty_oh.

  /* switch alias qaddb back to starting database */
  new_site = old_site.
  {gprun.i "gpalias.p"}

  /* set new_site back to user supplied site value */
  new_site = pt_site.
end.

```

gpalias.p requires the shared variable, new_site.

Accessing the database assigned the qaddb alias needs to be done in an external procedure.

Using gpalias: An Example

```
/* multi02.p - test multidb connection */  
  
define input parameter  part like pt_part no-undo.  
define input parameter  site like in_site no-undo.  
define output parameter qty_oh as decimal no-undo.  
  
for first in_mstr no-lock where in_part = part  
and in_site = site:  
    qty_oh = in_qty_oh.  
end.
```

Get the quantity on hand for the given part/site in the current database.



Multi-Database Considerations

▲ Share Common databases

- qadhelp
- qadadm

```
Exec $DLC/bin/_progres newyork -ld ny \  
-db chicago -ld ch \  
-db seattle -ld se \  
-db mfghelp -ld qadhelp \  
-db mfgadm -ld qadadm \  
...
```

Only include databases newyork, chicago, and seattle in the connection master. The other databases do not contain any “site” specific information. They can be shared with whatever database is active.

Summary



Multi-Database Programming

- ✓ Use of the Alias statement
- ✓ Use of Database functions
- ✓ Multi database code samples

Chapter 10

Compiling Code

Introduction



Compiling Code

In this section, you learn about:

- ▲ Logical assignment of the database name
- ▲ Empty database
- ▲ Custom source code - what to include
- ▲ Custom compile setup

Logical Database Names



QAD Enterprise Applications on PROGRESS Logical Database Names

Database	Physical	Logical
mfg (production)	mfg	<blank>
mfg (compile)	mfgempty	qaddb
help (production)	mfghelp	qadhelp
help (compile)	hlpempty	qadhelp
admin (production)	mfgadm	qadadm
admin (compile)	admempty	qadadm

Logical Assignment

```
exec $DLC/_progres mfgdemo  
-db mfghelp -ld qadhelp
```

The default logical name is the physical database name.

Since there is no **-ld** parameter following the connection of **mfgdemo**, the logical database name is "mfgdemo."

The logical database name for **mfghelp** is "qadhelp" since that is what is specified by the **-ld** parameter.



Logical Assignment

Compilation:

```
exec $DLC/_progres mfgempty -1 -crc -ld qaddb
      -elpcD 63 30 utcomp.p 75 75
      -db cstempty -1 -crc -ld qaddb_s
      -db hlpempty -1 -crc -ld qadhelp
      -db admempty -1 -crc -ld qadadm
```

The logical DB name QADDB should only be assigned during compile time.

At runtime, let mfgpro assign the alias QADDB

Runtime:

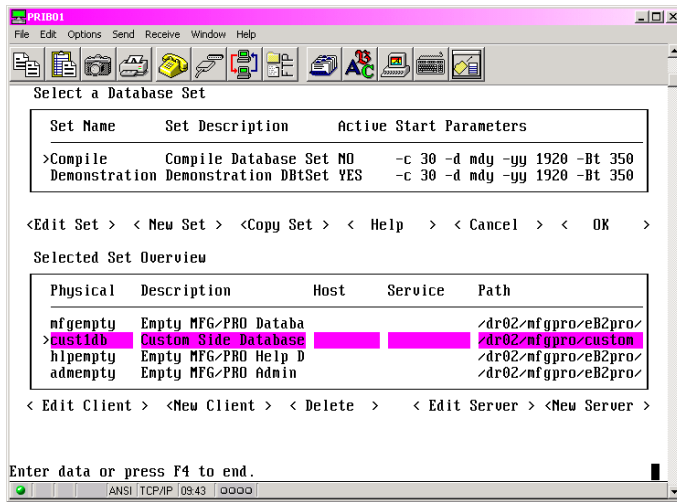
```
exec $DLC/_progres mfgdemo -ld demo
      -db qad_side -ld qaddb_s
      -db mfgghelp -ld qadhelp
      -db mfgadm -ld qadadm
```

NOTE: QAD Enterprise Applications startup programs assign the alias "qaddb" to the first connected database

Compiling Using MFG/UTIL

Compiling Using MFG/UTIL

- ▲ Configure
- ▲ Any Database Set



The compile database set must contain a complete set of "empty" QAD Enterprise Applications databases.

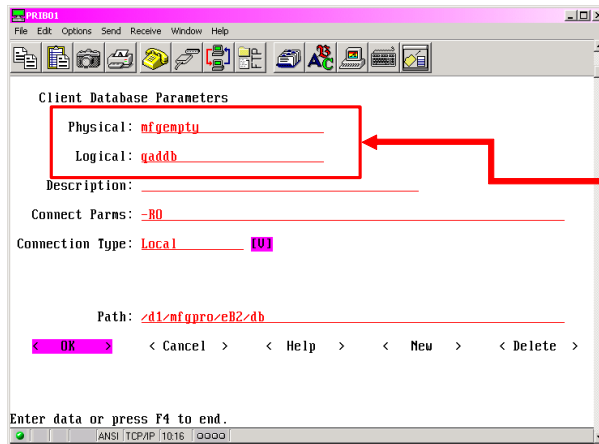
Add your empty custom database to this list for compiles.

Remember to assign a logical database name to each "client" entry.

Choose a "standard" logical name for your custom database.

Compiling Using MFG/UTIL

- ▲ Configure Any Database Set
 - Edit Client



Client Database Parameters

Physical: mfgempty

Logical: qaddb

Description: _____

Connect Params: -R0

Connection Type: Local

Path: /d1/mfgpro/eB2/db

< OK > < Cancel > < Help > < New > < Delete >

Enter data or press F4 to end.

The standard logical database names must be used for the compile set.

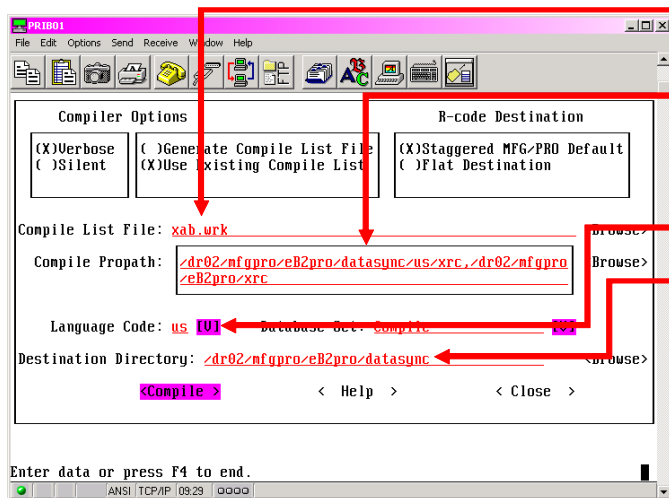
The QAD Enterprise Applications empty database (mfgempty) **MUST** have the logical database name of qaddb.

This name will be compiled into the r-code.



Compiling Using MFG/UTIL

▲ Program Menu — Compile Procedures



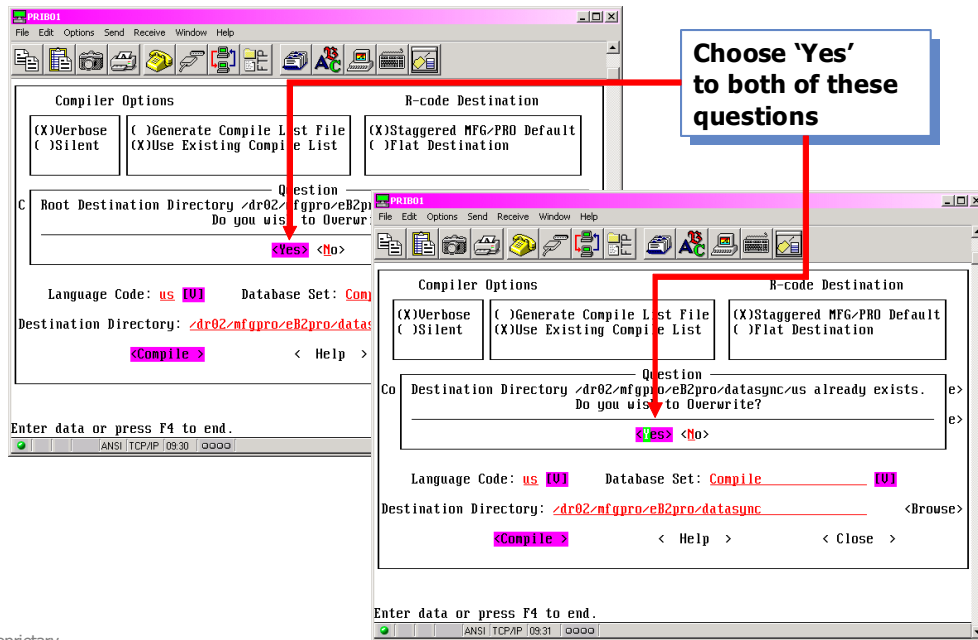
List of the procedures to be compiled.

Compile PROPATH is used for locating procedures in the Compile List file and their related include files.

Specify the language.

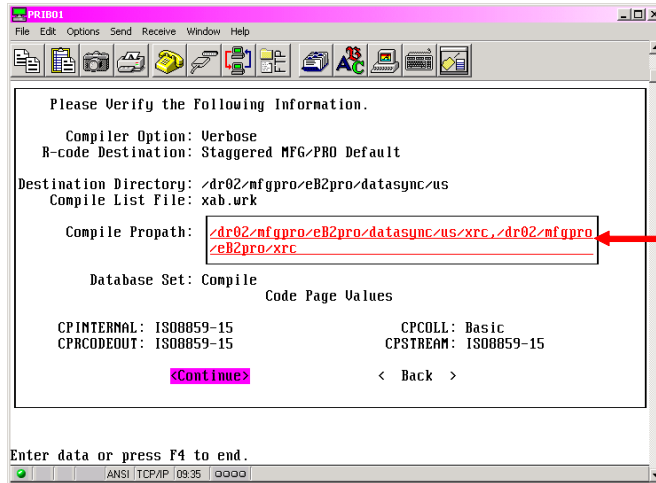
Specify the destination of the compile. R-code is automatically written in the two-letter subdirectory structure below the language within this base directory.

Compiling Using MFG/UTIL





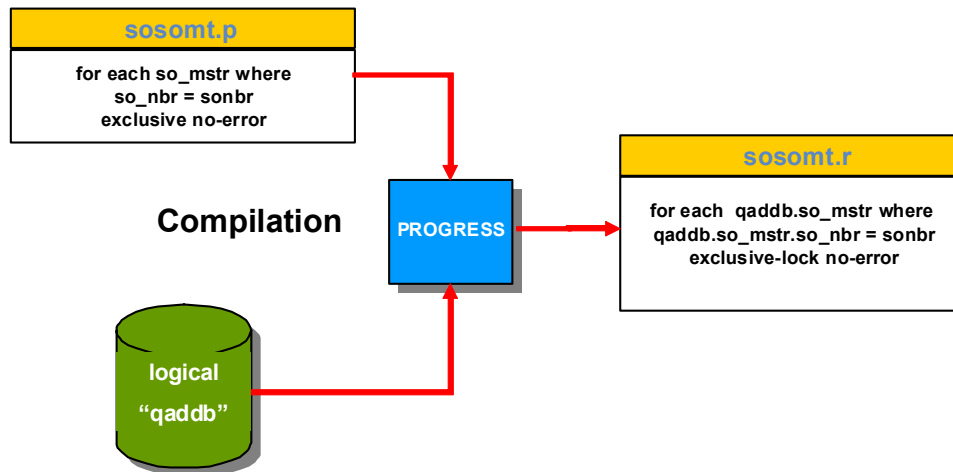
Compiling Using MFG/UTIL



One last chance
to change the
Propath!

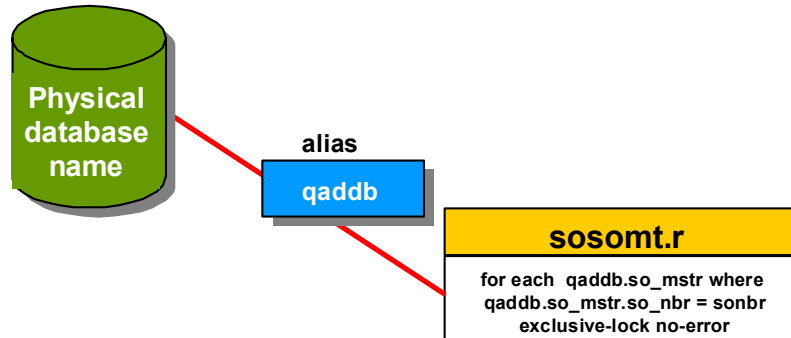
Logical Database Name at Compilation

- ▲ Data path stored in object code (.r)



Logical Database Name at Execution

Execution



General Custom Compile Setup

- ▲ Custom source code
 - Put in a separate “custom source” directory

```
mfgpro/  
    conv/  
    custom/  
        us/  
            custsrc/          c1sosomt.p...  
  
    mfg/  
    mfgdemo/  
    us/  
        src/  
        xrc/
```

Make sure all the programs (.p files) that need to be compiled for the custom source are included in this “custom source” directory.



General Custom Compile Setup

- ▲ Keep the compiled custom code separate from the standard QAD Enterprise Applications compiled code

```

mfgpro/
  conv/
    custom/
      .      us/
      .      .      cl/
      .      .      .      clsosomt.r...
      .      .      custsrc/
      .      .      .      clsosomt.p...
    us/
      .      ad/
      .      src/
  
```

Customizations

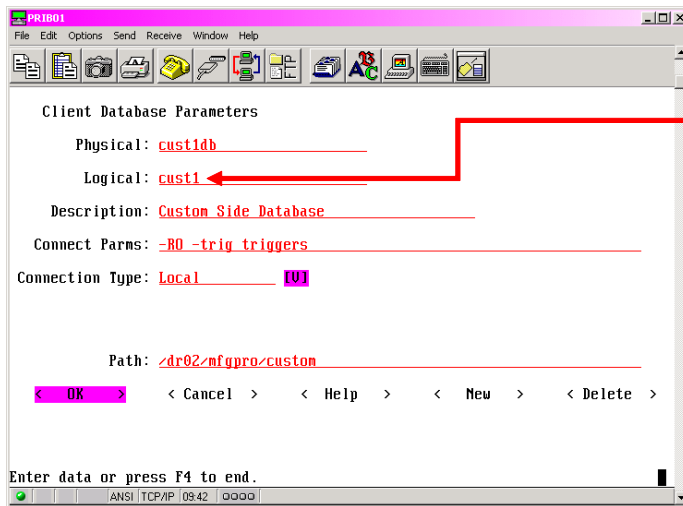
Standard QAD Enterprise Applications

Remember it is important to preserve the standard QAD Enterprise Applications environment.

Compile custom source into a custom directory structure.

General Custom Compile Setup

- ▲ Add custom side database to compile dataset

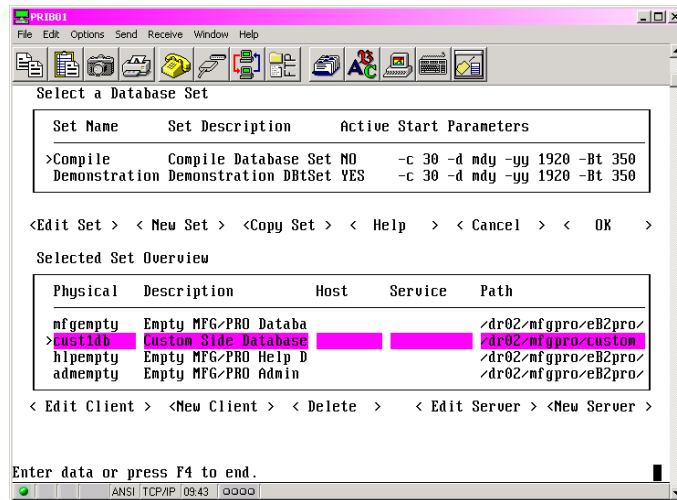


The logical database name "cust1" will be the logical database name we will always use to connect to the custom side database.



General Custom Compile Setup

▲ Complete compile dataset



General Custom Compile Setup

▲ Update startup scripts

- Add the custom code to the PROPATH
- Add the custom database to the connection statement

```
.  
#PROPATH=./dl/mfgpro:/dl/mfgpro/us/bbi  
PROPATH=./dl/mfgpro/custom:/dl/mfgpro:/dl/mfgpro/us/bbi  
export PROPATH  
  
exec $DLC/bin/_progres mfg -ld qaddb\  
-db custldb -ld cust1 \  
-db mfghelp -ld qadhelp \  
  
.  
.
```

Add the base custom directory to the beginning of the PROPATH.
Put right after the current working directory (.).

Hands On Exercise: Compiling Lab

- Compile your custom versions of Customer Maintenance (cladcsmt.p, cladcsmt.p, and cladcsmt02.i)
- Keep the custom environment separate from the standard QAD Enterprise Applications environment

Solution: Compiling

▲ Patch environment

- Use MFG/UTIL to compile programs and keep the QAD Enterprise Applications two-letter directory structure

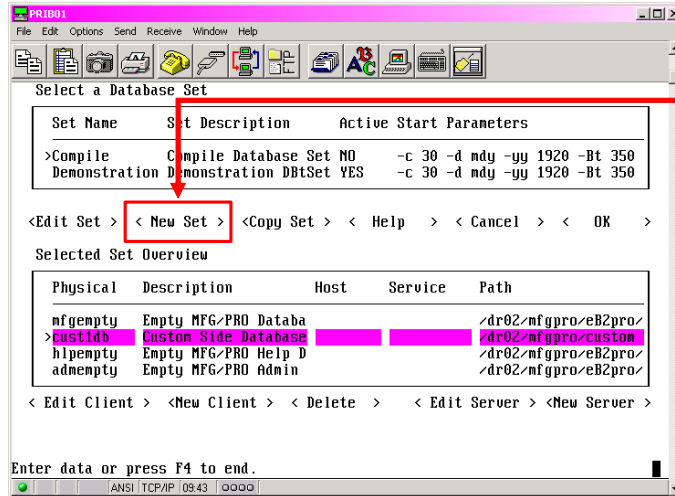
```
patch
patch/us
patch/us/src
patch/us/src/cladcsmt.p
patch/us/src/cladcsmt.p
patch/us/src/cladcsmt02.i
patch/us/c1
patch/us/c1/cladcsmt.p.r
patch/us/c1/cladcsmt.r
```



Solution: Compiling

▲ MFG/UTIL

– Configure, Any Database Set...

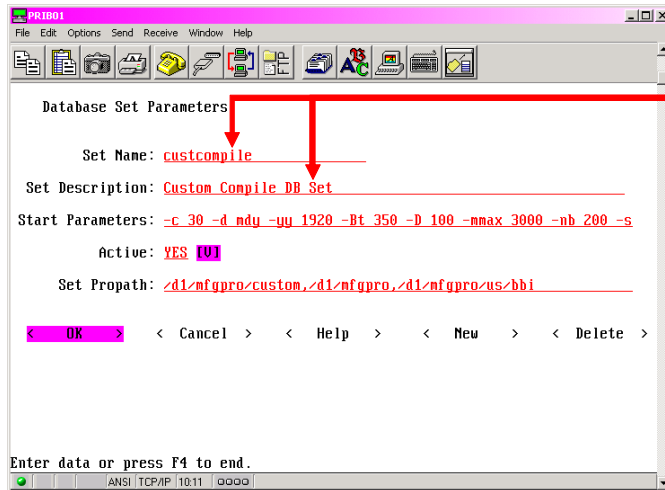


Select "New Set" to add a new Custom compile database set.

Solution: Compiling

▲ MFG/UTIL

- Creating a new database set



Provide a new set name and description.

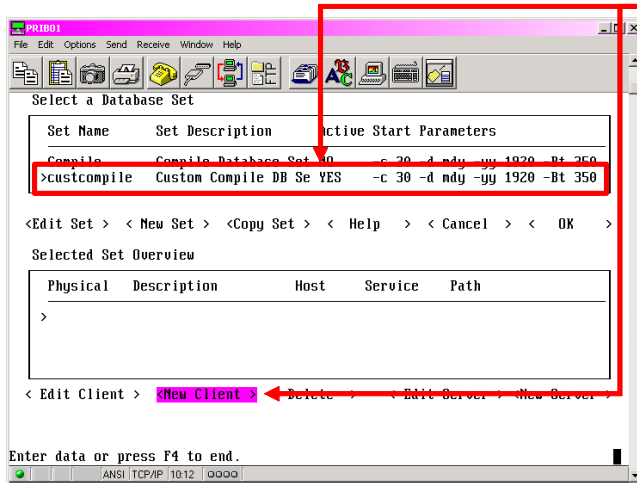
Keep the rest of the defaults.



Solution: Compiling

▲ MFG/UTIL

- Adding clients to the new database set



Select the new database set (use the arrow keys to select from the list of database sets).

Tab over to "New Client" to add the compile databases to your new database set.

You will need to create a client for the standard "compile" database set plus the new custom side database.

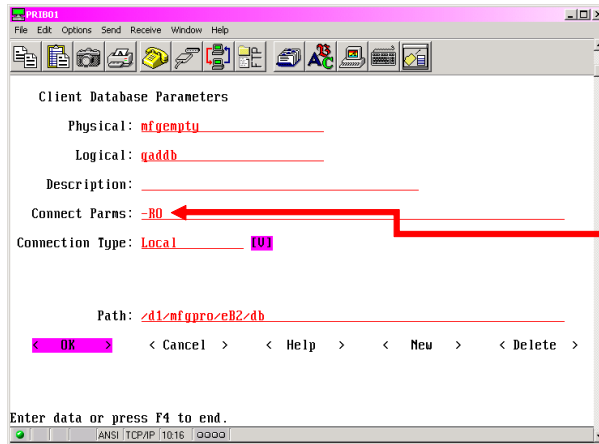
QAD Proprietary

BP-CRM-COM-220

Solution: Compiling

▲ MFG/UTIL

- Adding clients to the new database set



Client Database Parameters

Physical: mfgempty

Logical: qaddb

Description: _____

Connect Params: -R0

Connection Type: Local Y

Path: /d1/mfgpro/eB2/db

OK < Cancel > < Help > < New > < Delete >

Enter data or press F4 to end.

To enter the client, you need to provide the empty database to compile against, the logical database name that is needed to run QAD Enterprise Applications.

Connect to the database Read Only.

Make a "Local" connection.

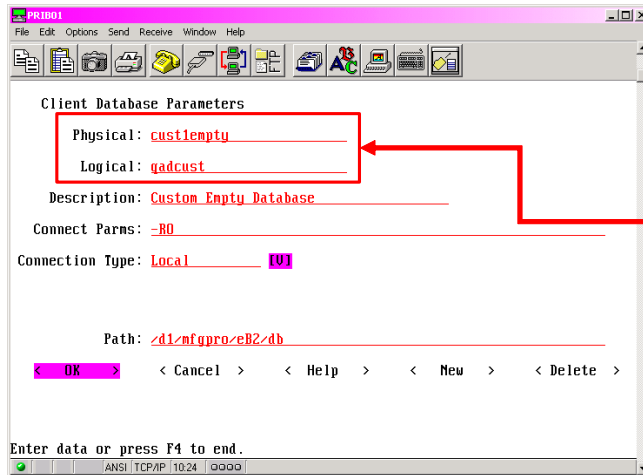
Enter the full path to the empty database.



Solution: Compiling

▲ MFG/UTIL

- Adding the custom side database to the new database set

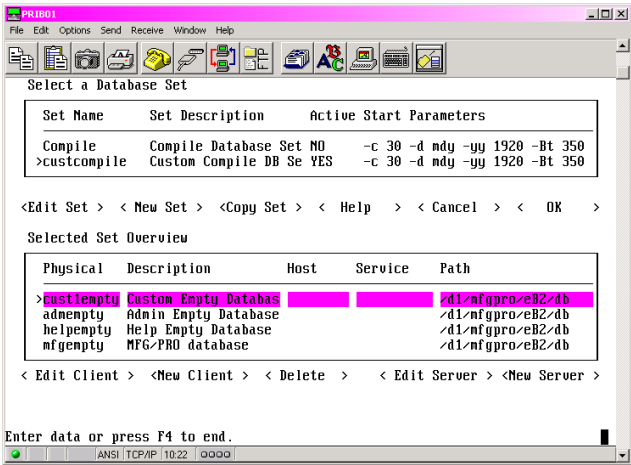


The custom side database should be created the same as any other database.

At this point, however, you need to choose the side database's logical name that will be used at compile time, as well as at run time.

Solution: Compiling

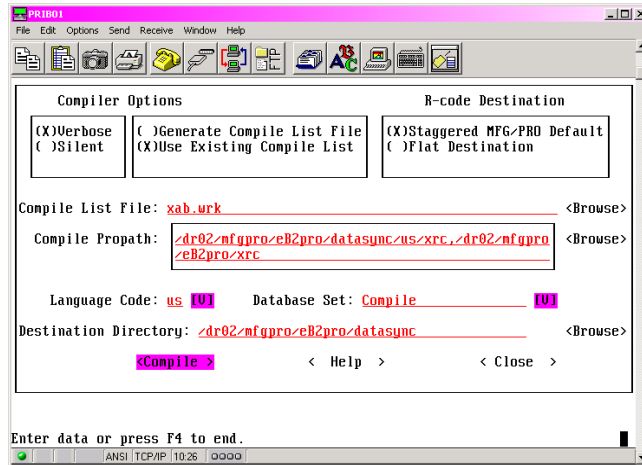
- ▲ MFG/UTIL
 - Completed custom compile database set





Solution: Compiling

- ▲ MFG/UTIL program menu
 - Compile Procedures



Will need to edit just about every entry point.

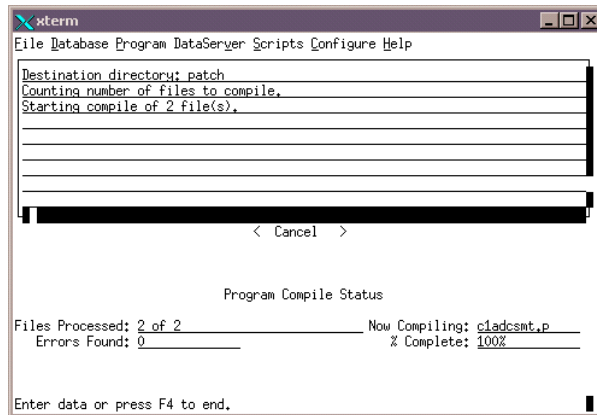
The Language field tells what language to store the subdirectories in.

The destination directory is the base directory that contains the language subdirectory.

Add the custom source directory to the front of the propath.

Solution: Compiling

- ▲ MFG/UTIL compile procedures function
 - Screen shows the compile status



Compile errors will be listed in the "editor" box at the top of the screen.

The errors are also written to the mfgutil.log file.



Solution: Compiling

- ▲ MFG/UTIL compile procedures function
 - Compile complete

```
xterm
File Database Program DataServer Scripts Configure Help

Destination directory: patch
Counting number of files to compile.
Starting compile of 2 file(s).

Compile contained 0 Errors.
Resetting PROPATH and disconnecting from databases.
End Compile: 10/12/98 13:31:51
press CLOSE to exit.

< Print > < Close >

Enter data or press F4 to end.
```

The number of errors is listed at the bottom of the editor frame.

You can tab to the editor and scroll up to see any errors that might have occurred.

Select "Close" when you are ready to exit.

Solution: Compiling

▲ Custom code directory structure...

```
/dr02/mfgpro/eB2pro >find patch -print  
patch  
patch/us  
patch/us/src  
patch/us/src/cladcsmt.p  
patch/us/src/cladcsmt.p  
patch/us/src/cladcsmt02.i  
patch/us/c1  
patch/us/c1/cladcsmt.p.r  
patch/us/c1/cladcsmt.r  
patch/triggers
```

Custom patched code was compiled into the standard QAD Enterprise Applications two-letter subdirectory environment within the "patch" subdirectory, which was specified in the MFG/UTIL compile.



Solution: Compiling

- ▲ Start-up script to include custom compile
 - Add the base custom directory to the PROPATH

```
.
.
PROPATH=/dr02/mfgpro/eB2pro/patch:./dr02/mfgpro/eB2pro:/dr
02/mfgpro/eB2pro/us/bbi;export PROPATH

# Start QAD Enterprise Applications.

exec $DLC/bin/_progres \
/dr02/mfgpro/eB2pro/mfgdemo -trig triggers -1 \
-db /dr02/mfgpro/eB2pro/mfghelp -ld qadhelp -1 \
-db /dr02/mfgpro/eB2pro/mfgadm -ld qadadm -1 \
-db /dr02/mfgpro/eB2pro/mfgcust -ld qadcust -1 \
-cpstream ibm850 -cpinternal iso8859-1 \
-cprcodein iso8859-1 -cpcoll basic \
-Bt 350 -c 30 -D 100 -mmax 3000 -nb 200 -s 48 -p mf.p;
```




Solution: A Recap

QAD Enterprise Applications Directory

US/SRC

/XRC

US/AD

.

.

/MR

.

.

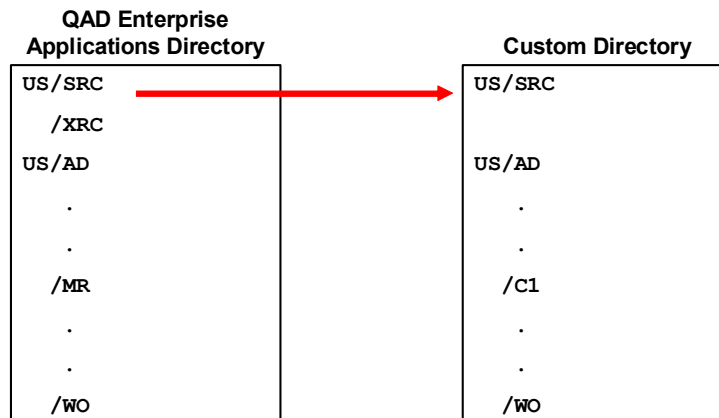
/WO

Compile: PROPATH=.:\$MFGPRO/us/xrc:\$MFGPRO/us/bbi

Runtime: PROPATH=.:\$MFGPRODIR:\$MFGPRO/us/bbi



Solution: A Recap



Compile: `PROPATH=$CUSTMFGPRO/us/src:.$MFGPRO/us/xrc:$MFGPRO/us/bbi`

Runtime: `PROPATH=$CUSTMFGPRO:.$MFGPRODIR:$MFGPRO/us/bbi`

Solution: Additional Exercise

The screenshot shows the 'Menu System Maintenance' window. On the left is a tree view of the menu structure. The path 'Master Data > Addresses / Taxes > Customers Menu > Customer Maintenance' is selected. The right pane shows the configuration for this menu item:

- Type: .NET UI Menu
- Language: us
- Label: Customer Maintenance
- Image: Maintenance
- Exec Procedure: adcsmt.p
- Name: cust
- Help File: adcsmt.hlp
- Menu: 2.1
- Selection: 1

A callout box on the right contains the following text:

In Menu System Maintenance, change the Selection Label to indicate that the menu has changed.

Replace the standard program adcsmt.p with the custom program c1adcsmt.p.

Menu System Maintenance (36.4.4)



Solution: Additional Exercise

- ▲ Add program to Program Information Maintenance to enable it to work in Desktop 2
- ▲ Rebuild the Desktop

Program Information Maintenance

Go To Actions Copy Print Preview

Program: c1adcsmt

Selection Label:

Multi Domain ☐

Web Options

Image:

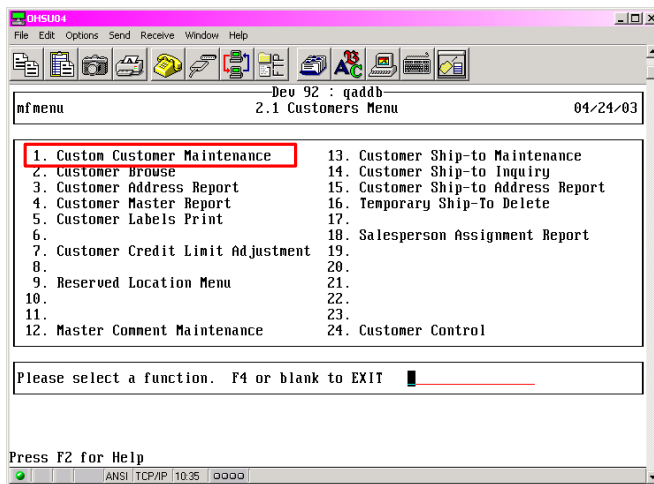
Web Logic Implemented: ☒

Type: desktop

Program Information Maintenance (36.3.21.1)

Solution: Additional Exercise

▲ Look at the Customers Menu



**In Customers Menu...,
the menu label has
been changed.**



Solution: Additional Exercise

- ▲ Run Custom Customer Maintenance from the 2.1.1 menu selection

Customer Address

Customer: Name:

Address:

Address:

Address:

City: State: Post: Format:

Country: County:

Attention: [2]:

Telephone: Ext: [2]:

Fax/Telex: [2]:

E-Mail: [2]:

Customer Data

Sort Name: Type: Taxable:

Salespsn1: Multiple: Region: Price Tbl:

Ship Via: Currency: Disc Tbl:

AR Acct: Language: Fixed Price:

Resale: Site: Class:

Remarks: Partial OK:

F1=Go 2=Help 3=Ins 4=End 6=Menu 7=Recall 8=Clear 9=Prev 10=Next

All the modifications come up when menu selection 2.1.1 is run.

Review



Compiling Code

- ✓ Logical assignment of the database name
- ✓ Empty database
- ✓ Custom source code - what to include
- ✓ Custom compile setup

Chapter 11

Triggers

Introduction



Triggers

In this section, you learn about:

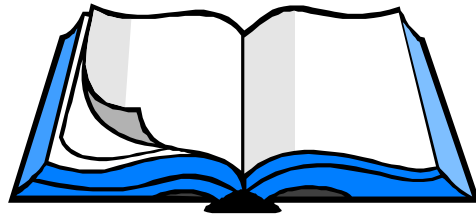
- ▲ Procedure statement
- ▲ Headers
 - Create
 - Delete
 - Find
 - Write
 - Assign
- ▲ Return errors

Terminology



Terminology

- ▲ Trigger
- ▲ Metaschema



QAD Proprietary

BPC-11-TRI-024

Trigger. A piece of code that Progress executes on behalf of an application when a certain database action occurs or a certain user interface event occurs.

Metaschema. Tables or fields with initial underscore.

Schema Triggers



Schema Triggers

- ▲ Create schema triggers through the table or field properties dialog box in the data dictionary
- ▲ The trigger is automatically added to the table or field's data definition
- ▲ A trigger procedure is a PROGRESS procedure file that begins with a trigger header statement
- ▲ Use the Triggers (-trig) startup parameter to specify the procedure library that contains them, otherwise PROGRESS searches the PROPATH for the trigger procedure (.t)

QAD Proprietary

BPC-11-TRI-03

Launch the dictionary in single user mode in order to create a schema trigger.

QAD uses the -trig parameter in the default start-up scripts. Can also compile triggers into a procedure library and point to the library with the -trig parameter.

Trigger Procedure Statement

▲ Syntax

TRIGGER PROCEDURE FOR *event* OF *object* [*options*]

- *event*

The event for which the schema trigger is being defined. The valid events are CREATE, DELETE, FIND, WRITE and ASSIGN

- *object*

The object on which the event is defined. If the event is CREATE, DELETE, FIND, or WRITE, the object must be a reference to a database table. If the event is ASSIGN, the object must be a reference to a database field qualified by a table name

- *options*

Optional parts of the trigger header

CREATE, DELETE, FIND Headers



CREATE, DELETE, & FIND Headers

▲ Syntax

TRIGGER PROCEDURE FOR
table

{
 FIND
 CREATE
 DELETE
 REPLICATION-CREATE
 REPLICATION-DELETE
 }

OF

- References to the record can be made through an automatic buffer defined as part of the trigger header
- Its name is the same as the table name

CREATE Trigger Header

▲ Basic Syntax

TRIGGER PROCEDURE FOR {
CREATE
REPLICATION-CREATE } OF *table*

▲ Order of execution

- Executes a CREATE statement
- Creates record
- Fires all applicable CREATE triggers
- Fires all applicable REPLICATION-CREATE triggers



DELETE Trigger Header

▲ Basic Syntax

TRIGGER PROCEDURE FOR
table {
 DELETE
 REPLICATION-DELETE } OF

▲ Order of execution

- Executes a DELETE statement
- Fires all applicable DELETE triggers
- Fires all applicable REPLICATION-DELETE triggers
- Validates the DELETE
- Performs the DELETE

FIND Trigger Header

▲ Basic Syntax

TRIGGER PROCEDURE FOR **FIND** OF *table*

▲ Order of execution

- Executes a FIND or FOR EACH or GET statement
- Fires all applicable FIND triggers
 - Only fires for those records that completely satisfy the full search condition
 - Does not fire in response to the CAN-FIND function
 - If a find trigger fails, the record select fails as well

WRITE Trigger



WRITE Header

▲ Syntax

```

TRIGGER PROCEDURE FOR { WRITE
                        REPLICATION-WRITE } OF table
[NEW [BUFFER] buffer-name1] [OLD [BUFFER] buffer-name2]

```

- The NEW buffer contains the modified record that is being validated
- The OLD buffer contains the most recent version of the record before the latest set of changes were made
- The contents of the NEW buffer can be modified but the OLD buffer is read-only

QAD Proprietary

BPC-11-TRI-09

You have access to the old record buffer and the new one, allowing for good comparison validations.

WRITE Header

▲ Syntax

```
TRIGGER PROCEDURE FOR { WRITE  
                        REPLICATION-WRITE } OF table  
[NEW [BUFFER] buffer-name1] [OLD [BUFFER] buffer-name2]
```

▲ Order of Execution

- Record contents change is executed
- Fires all applicable WRITE triggers
- Fires all applicable REPLICATION-WRITE triggers
- Record contents are changed
- Record changes are written



WRITE Trigger Example

```
TRIGGER PROCEDURE FOR WRITE OF cm_mstr OLD BUFFER oldcust
.

define variable ordcnt as integer no-undo initial 0.

/* check to see if the user changed the cm_addr field */
if cm_mstr.cm_addr <> oldcust.cm_addr then do:
    for each so_mstr where so_mstr.so_cust =
        cm_mstr.cm_addr
        exclusive-lock:
            so_mstr.so_cust = cm_mstr.cm_addr.
            ordcnt = ordcnt + 1.
        end.
    if ordcnt > 0 then
        message "Orders changed to reflect new customer
        information".
    end.
```

Assign Trigger



ASSIGN Header

▲ Syntax

TRIGGER PROCEDURE FOR ASSIGN

$\left\{ \begin{array}{l} \text{OF } table.field \\ \text{NEW [VALUE] } parameter1 \end{array} \right\}$	$\left\{ \begin{array}{l} \text{AS data-type} \\ \text{LIKE field} \end{array} \right\}$	$\left\{ \begin{array}{l} \text{COLUMN-LABEL } label \\ \text{FORMAT string} \\ \text{INITIAL constant} \\ \text{LABEL string} \\ \text{NO-UNDO} \end{array} \right\}$
$\left\{ \begin{array}{l} \text{OLD [VALUE] } parameter2 \end{array} \right\}$	$\left\{ \begin{array}{l} \text{AS data-type} \\ \text{LIKE field} \end{array} \right\}$	$\left\{ \begin{array}{l} \text{COLUMN-LABEL } label \\ \text{FORMAT string} \\ \text{INITIAL constant} \\ \text{LABEL string} \\ \text{NO-UNDO} \end{array} \right\}$



ASSIGN Header

- ▲ Access the value stored in the database by using the OLD option
- ▲ The value of the field before the assignment is stored in parameter2
- ▲ Use the NEW and OLD options to compare the value being written to the existing value
- ▲ PROGRESS treats the old value as an input run time parameter

ASSIGN Trigger Example

```
TRIGGER PROCEDURE FOR ASSIGN OF cm_mstr.cm_addr
    OLD VALUE oldaddr.
for each so_mstr where so_mstr.so_cust = oldaddr exclusive-
lock:
    so_mstr.so_cust = cm_mstr.cm_addr.
end.
```

- ▲ If you enable a WRITE trigger on a table and an ASSIGN trigger on a field within that table, the ASSIGN trigger always executes before the WRITE trigger
- ▲ If the WRITE trigger procedure assigns a value to the trigger field, this re-invokes the ASSIGN trigger

Return Error



Return Error

- ▲ If you use the NO-ERROR qualifier on a statement that executes a trigger, and the trigger returns an error, the ERROR-STATUS system handle is set to TRUE
- ▲ Check the value of this system handle after the statement executes

<pre>/* main.p */ delete cm_mstr NO-ERROR. if ERROR-STATUS:ERROR then do: then do: /* logic when error returned */ Credit Limit!". end.</pre>	<pre>/* delcust.t */ TRIGGER PROCEDURE FOR DELETE OF cm_mstr. if cm_balance > cm_cr_limit message "Balance exceeds RETURN ERROR. end.</pre>

General Considerations



General Considerations

- ▲ PROGRESS does not allow any database triggers on events involving the metaschema
 - metaschema are tables or fields with initial underscore
- ▲ Avoid statements that call for user input in a database trigger block
 - If the user runs the procedure in batch mode, a trigger with a prompt causes the procedure to stop, waiting for user input
- ▲ You cannot advance any record in a buffer passed to a database trigger procedure (as with a FIND NEXT or FIND PREV) within the trigger procedure



General Considerations

- ▲ For all blocks in a database trigger, the default ON ERROR phrase is ON ERROR UNDO, RETURN ERROR
 - This is in contrast to the usual PROGRESS default: ON ERROR UNDO, RETRY

Review



Triggers

- ✓ Procedure statement
- ✓ Headers
 - ✓ Create
 - ✓ Delete
 - ✓ Find
 - ✓ Write
 - ✓ Assign
- ✓ Return errors

Chapter 12

Browses

Introduction



Browses

In this section, you learn about:

- ▲ Look-up browse
- ▲ Power browse
- ▲ Views
- ▲ Tables
- ▲ Designing browses
- ▲ Printing browses
- ▲ Graphing browses

Look-up Browses



Look-Up Browses

- ▲ Displays a query associated with a field
- ▲ Replaces the scrolling windows
- ▲ Returns the value selected to the active field calling the browse
- ▲ Displays and sorts up to 7 columns of fields
- ▲ Must be linked to a field
- ▲ Can be accessed by:
 - Clicking on lookup icon
 - Clicking on the Help menu, choose Look-Up Browse
 - Pressing ALT-F2 (press F2 in character)

QAD Proprietary

BPC-12-BRO-020

NetUI has a Look-up icon and the Help menu access.

Power Browse



Our Passion. Your Advantage.

Power Browse (1 of 2)

- ▲ Features the full functionality for sorting, filtering, graphing, printing, and exporting data
- ▲ Displays up to 20 columns of fields
- ▲ The first 7 columns can be sorted
- ▲ The first 10 columns can be filtered

QAD Proprietary

BPC-12-BRO-030

Replaces QAD Enterprise Applications inquiry programs. Usually associated with a menu number.

NetUI allows sorting on all available fields.

Power Browse (2 of 2)

- ▲ Links to a field or runs independently
- ▲ Can be accessed by:
 - Pressing ALT-F1
 - Clicking on the Help Menu and choosing Drill Down
 - Assigning the browse a menu number and running it
 - Putting a browse on the toolbar as a button
 - Embedding it in an OBCM maintenance application as a folder

Views



Views

- ▲ A display of all fields and locally defined variables from one or more tables
- ▲ Can join two or more tables by specifying the relationship (the linking comparisons) between fields in the various tables
- ▲ Used in browses to display the data gathered using the view

QAD Proprietary

BPC-12-BRO-055

Joins in views are inner joins only (table intersects).

Browse Tables in GUI Database



Browse Tables in the Adm DB (1 of 2)

- ▲ brw_mstr - Browse Master
 - Defines a Browse
 - includes name, description, where clause, browse type
- ▲ brwf_det - Browse Field Definitions
 - Defines the fields that are used in each browse
 - includes field names, format, order
- ▲ brwt_det - Browse Table Definitions
 - Defines the join from the view
 - includes table names and join criteria
- ▲ vue_mstr - View Master
 - Defines a View
 - includes name, description, password

QAD Proprietary

BPC-12-BRO-066

This is the reason that the gui database is required for both the GUI and Character environments.



Browse Tables in the Adm DB (2 of 2)

- ▲ vuf_det - View Field Definitions
 - Defines the field definitions used in the view
 - includes field names, format, expression
- ▲ vwj_det - View Table Definitions
 - Defines the view join criteria
 - includes table name, order, join criteria
- ▲ drl_mstr - Drill Down Master
 - Attaches “Power Browse” to a field to make a “Drill Down”
 - includes field, calling program, executable
- ▲ mnts_det - Menu Translation Table
 - All fields from Browsers, Views, Drill Downs which have labels to be translated. Example: brw_desc, brwf_label, vuf_label

Designing Custom Browses



Designing Browses (1 of 2)

- ▲ Construct the browse in a manner that maximizes the retrieval rate of records
- ▲ The first field should be indexed and should be from the first table in the view
- ▲ Local variables cannot be the first field in a browse
 - This will work with Look-Up Browses, but not Power Browses so we recommend that it not be done at all

QAD Proprietary

BPC-12-BRO-085

Consider sort fields and indexes.



Designing Browsers (2 of 2)

- ▲ All of our views are inner joins; that is, if the record in the second table is not found, the record from the first table is not included in the browse
 - Do not create views for relationships where there may not be related data in the second table
 - Instead, create one browse for the first table and then create another for the second table and connect the two browses via Drill-Down Maintenance

Naming Conventions



Naming Conventions

- ▲ View names use the first table name then remove the underscore and append a unique numerical sequence:

<u>Table Name</u>	<u>View Name</u>
ar_mstr	armstr01,armstr02,.....
sod_det	soddet01, soddet02,....

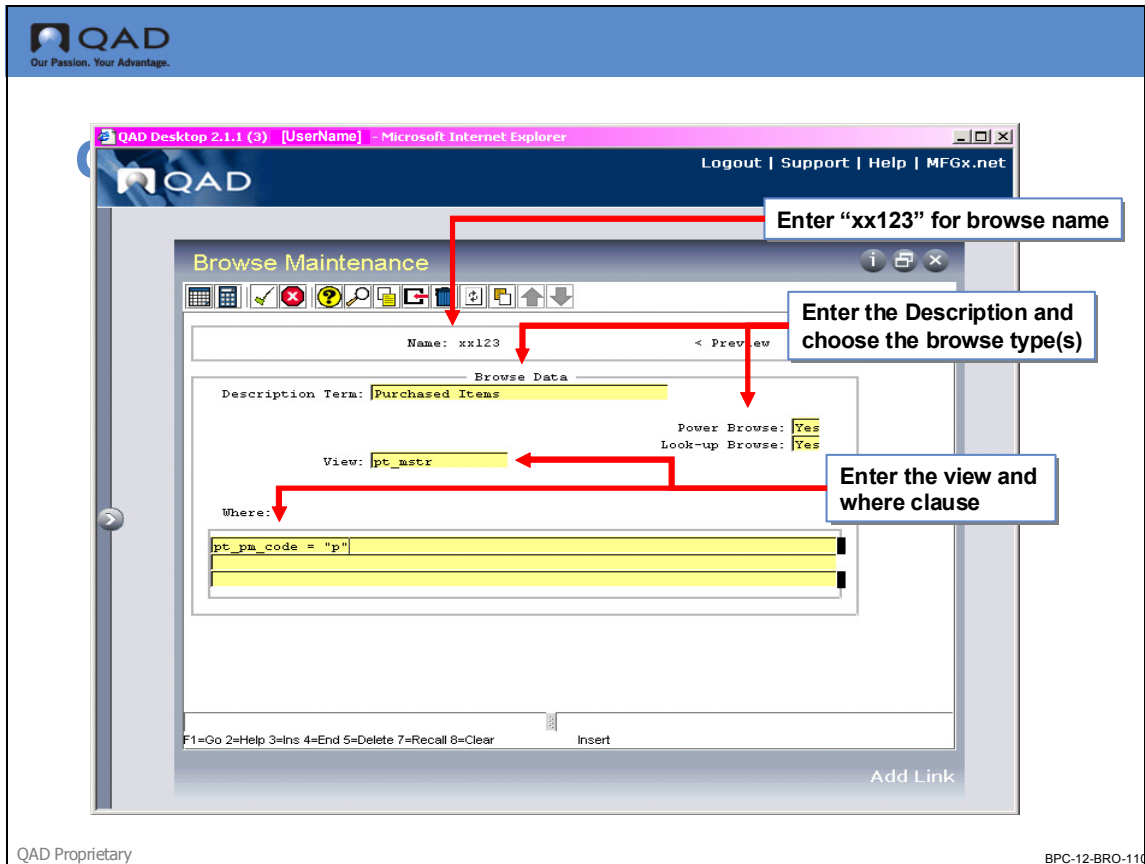
- ▲ Browse names refer to the appropriate module (for example, gp for General Purpose) followed by a three-digit unique sequence

<u>Browse Name</u>	<u>Source File Name</u>
ar001	arbr001.p, arlu001.p
gp367	gpbr367.p,gplu367.p

QAD Proprietary

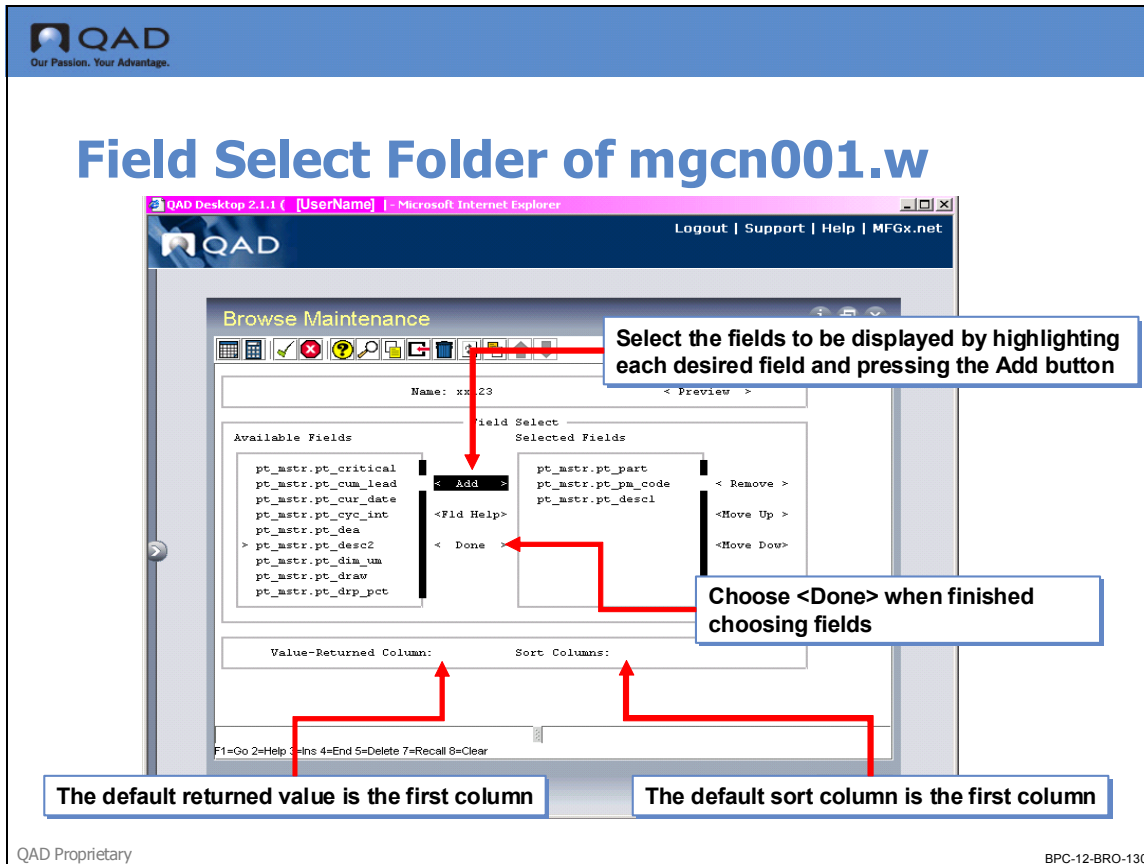
BPC-12-BRO-100

BR or LU is automatically added to the filename by the Browse Maintenance programs. If you specify both, then two programs are generated.



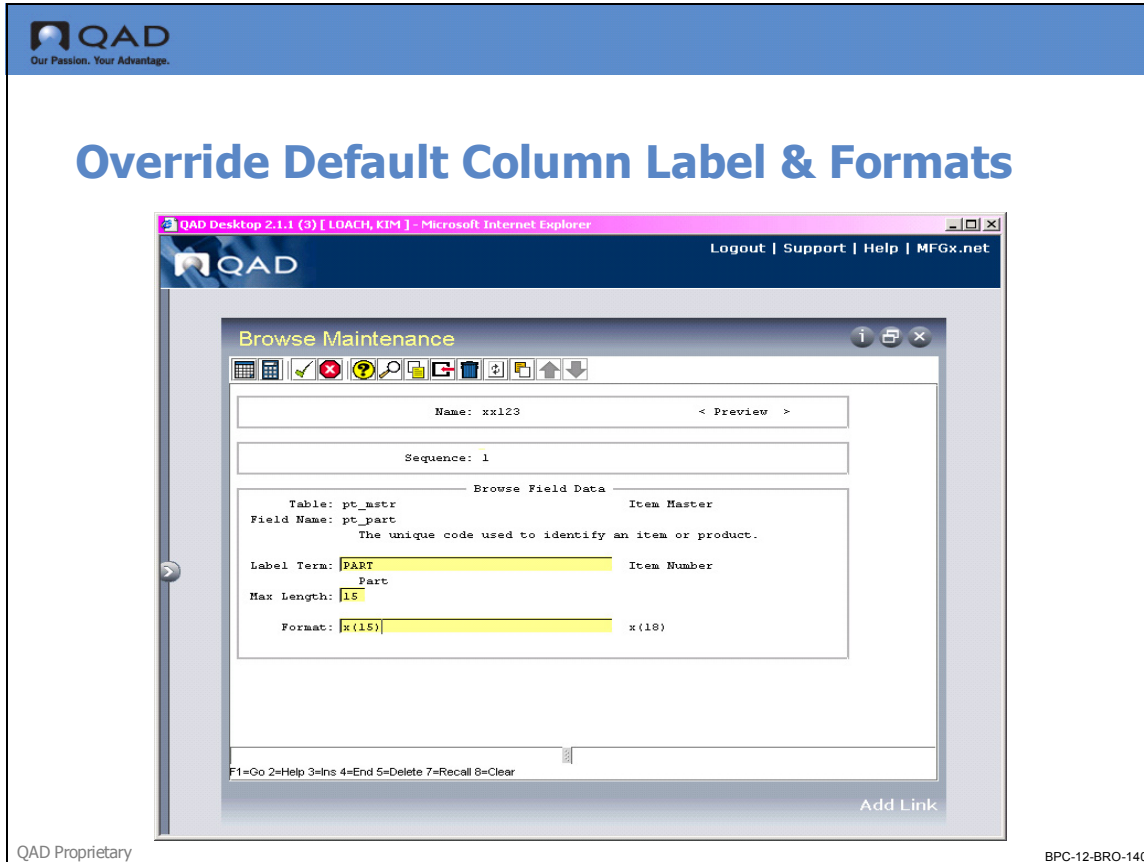
Browse Data Folder of mgcn001.w

- ▲ Enter the name of the browse you are creating
 - We encourage our customers to use xx for custom browses. This will eliminate conflicts with browses created by QAD.
- ▲ Browse type
 - Can select both Power Browse and Look-up Browse

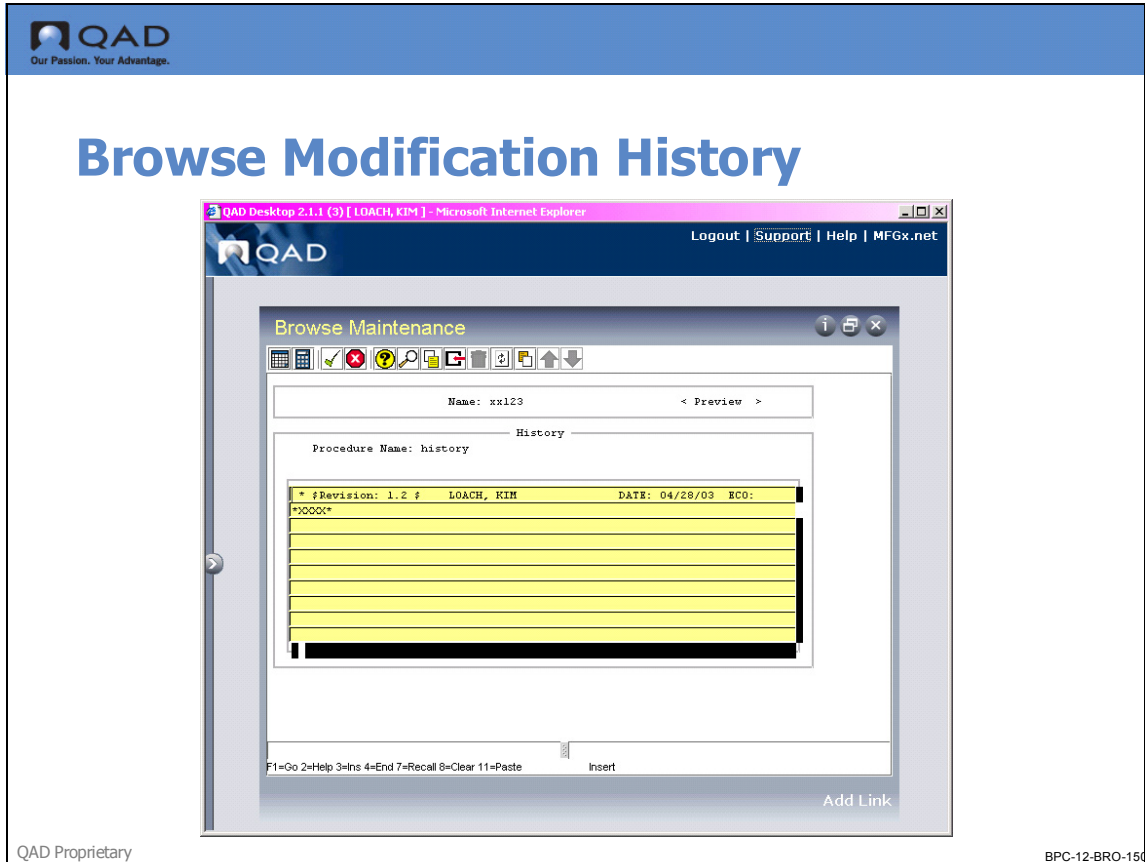


Functional, but awkward in Character mode.

Value returned is the column from which you want the data to return to the interface.



In Character, access this screen by pressing Esc-M in the Browse Maintenance screen. This accesses the menus. In the Window menu select Fields.



QAD Proprietary

BPC-12-BRO-150

Browse Modification History

- ▲ This screen is used to record the modification log of the browse you created
 - An entry should be made when the browse is created and every time the browse is modified
 - This historical data gets transcribed on the PROGRESS code created from Browse Maintenance

Generated Source Code

- ▲ The browse generator creates the following source code:

<u>Browse Name</u>	<u>PROGRESS Source Code Created</u>	<u>Type of File</u>
xx123	xxbr123	Power Browse
	xxlu123	Look-Up Browse

- ▲ The names of the files are based on the name of the browse given
- ▲ They are created in the current working directory

Generated Source Code



Linking the Look-Up Browse (1 of 2)

- ▲ In most cases (all cases for Look-Ups), the intended use of browses is to have them launched from a specified field on a screen
 - Let's link the browse to Purchase Order Maintenance at the point the user enters line item numbers
 - Specifically when the user is on the line item number field

Linking Browsers



Linking the Look-Up Browse (2 of 2)

- ▲ The xxlu123.p and xxbr123.p programs must be moved from the working directory to the custom source directory
 - For our example they were moved to c:\custom\us\src
 - Use MFGUTIL to compile these programs into the standard staggered structure

QAD
Our Passion. Your Advantage.

Popomt.p

Purchase Order Maintenance

Go To Actions Copy Print Preview

Header Lines Trailer

Lines Line Details Tax Info Comments

Header

Purchase Order: PD12345 Supplier: 5001000 Ln Format S/M: Single

Lines

Ln	Site	Req	Item Number	Qty Ordered	UM	Unit Cost	Disc%
2	train						

Line Details

Qty Received: Due Date: CRT Int:

Qty to Rel: Pur Acct:

Single Lot: ☐ Performance Date: Project:

Location: Need Date: Type:

Revision: Sales/Job: Taxable: ☐

Status: Fixed Price: ☐ Inspection Requi: ☐ Cmmts: ☐

Supplier Item: UM Conversion:

Manufacturer: Stock UM Quantity:

Description: Update Avg/Last Cost: ☐

Extended Net Cost:

Back Next

This is the field where we want to launch our browse

QAD Proprietary Purchase Order Maintenance (5.7) BPC-12-BRO-200

QAD
Our Passion. Your Advantage.

Drill/Down Lookup Maintenance

The screenshot shows the 'Drill Down/Lookup Maintenance' window. It has a title bar with a close button. Below the title bar is a menu bar with 'Go To', 'Actions', 'Copy', 'Print', and 'Preview'. The main area contains the following fields and labels:

- Drill Down/Lookup:** Lookup
- Field Name:** pod_part
- Calling Procedure:** popomt.p
- Procedure To Execute:** xxlu123.p
- Description Term:**
- Description:** Item Number
- Lookup Starts At Row:** 7
- Lines In Lookup:** 6

Red arrows point from callout boxes to the corresponding fields:

- From **pod_part is the field** to **Field Name: pod_part**
- From **popomt.p is the calling procedure** to **Calling Procedure: popomt.p**
- From **xxlu123.p is the procedure to execute** to **Procedure To Execute: xxlu123.p**

At the bottom left, there is a link: [Adding new record](#). At the bottom right, there are **Back** and **Next** buttons.

QAD Proprietary BPC-12-BRO-210

Leave calling procedure blank to link the browse to all instances of the field in the interface.

QAD
Our Passion. Your Advantage.

Mgcn001.w - Link Power Browse

The screenshot shows a software window titled "Drill Down/Lookup Maintenance". It has a menu bar with "Go To", "Actions", "Copy", "Print", and "Preview". The main content area is divided into two sections. The top section contains the following fields:

- Drill Down/Lookup: Drill Down
- Field Name: pod_part
- Calling Procedure: popomt.p
- Procedure To Execute: xxbr123.p

Red arrows point from callout boxes to these fields:

- From "pod_part is the field" to the Field Name field.
- From "calling procedure is popomt.p" to the Calling Procedure field.
- From "procedure to execute is xxbr123.p" to the Procedure To Execute field.

The bottom section is titled "Description Term:" and contains:

- Description: Item Number
- Lookup Starts At Row: 7
- Lines In Lookup: 6

At the bottom right of the window are "Back" and "Next" buttons.

Drill/Down Lookup Maintenance (36.20.1)

QAD Proprietary BPC-12-BRO-220

Designing Views



Designing a View (1 of 2)

- ▲ Suppose you wanted to create a browse that displays the following inventory information
 - Item Number
 - Location
 - On-Hand Quantity
- ▲ This can be accomplished by selecting the Inventory Detail (Id_det) table when creating your browse
 - But what if the user wanted:
 - To display a calculated field that is not stored on the table
 - Information that is not on the Inventory Detail table, such as Item Description?

Designing a View (2 of 2)

- ▲ This can be accomplished by creating a view that joins the following two tables and defines a local variable for Qty Unallocated
 - Id_det - Inventory Detail (by Location)
 - fields needed: Id_part, Id_loc, Id_qty_oh, Id_qty_all
 - pt_mstr - Item Master
 - fields needed: pt_part, pt_desc1
 - Local variable: **local-var01** = Id_qty_oh - Id_qty_all

QAD
Our Passion. Your Advantage.

View Maintenance - mgcn008.w

View Maintenance

Go To Actions Copy Print Preview

Name: inv001

View/Join Data

Description Term: Inventory Unallocated
Enable QXtend Outbound

User IDs/Groups:

Enter the view name "inv001"

Enter the description "Inventory Unallocated"

Delete Back Next

View Maintenance (36.20.18)

QAD Proprietary BPC-12-BRO-250

QAD
Our Passion. Your Advantage.

If Label Does Not Exist

The screenshot shows the 'View Maintenance' window for 'inv001'. The 'View/Join Data' section displays 'Description Term: INVENTORY_UNALLOCATED' and 'Enable Qxtend Outbound'. A red arrow points from a text box to a warning dialog box that says 'WARNING: Label term does not exist. Create.' with 'Yes' and 'No' buttons. The 'Back' and 'Next' buttons are at the bottom right.

The Translation system checks if the Description Term exists and allows you to create it if necessary.

View Maintenance (36.20.18)

QAD Proprietary BPC-12-BRO-260

QAD
Our Passion. Your Advantage.

Adding label

View Maintenance x

Go To Actions Copy Print Preview

Name: inv001

Language: us
Term: INVENTORY_UNALLOCATED

Long Label: Unallocated Inventory

Medium Label:

Short Label:

Stacked Label:

Description: Custom View for Unallocated Inventory

Add the Label information and a description.
More on this in the chapter on Translations and Multi-language labels later in the course.

Delete Back Next

View Maintenance (36.20.18)

QAD Proprietary BPC-12-BRO-270

QAD
Our Passion. Your Advantage.

Table Selection Container

View Maintenance X

Go To Actions Copy Print Preview

Name: inv001

Sequence: 1

Table Selection

Table: pt_mstr

Join Type:

Enter one of the tables in the join: pt_mstr

Join Phrase:

Press Next again

Delete Back Next

QAD Proprietary BPC-12-BRO-280

In Character, access this screen by pressing Esc-M in the View Maintenance screen. This accesses the menus. In the Window menu, select Tables.

The screenshot shows the 'View Maintenance' window for 'inv001'. The 'Table Selection' section is highlighted. The 'Table' field contains 'Id_det'. The 'Join Type' dropdown is set to 'Inner'. The 'Join Phrase' field contains 'Id_part = pt_part'. Red arrows point from text boxes to these fields: 'Enter the other required table: Id_det' points to the 'Table' field, 'Choose the join type' points to the 'Join Type' dropdown, and 'Join the two table with the phrase: Id_part = pt_part' points to the 'Join Phrase' field. The window has a toolbar with 'Go To', 'Actions', 'Copy', 'Print', and 'Preview'. At the bottom, there are 'Delete', 'Back', and 'Next' buttons. The footer includes 'QAD Proprietary', 'View Maintenance (36.20.18)', and 'BPC-12-BRO-290'.

Inner (the default)

An inner join returns the records selected for the first table combined with related records selected from the second table. If a record does not exist in the second table, no records are returned. Only related records selected from both sides of the relationship display in the view.

Outer

An outer join returns the records found by an inner join. However, in addition, for each value in the first table, it returns unknown values from the second table when no related record is found. As a result, all matching records from the first table are preserved for unmatched records in the second table.

Using the outer join can give you more flexibility in displaying information. For example, an inner join between customers and sales orders displays only customers with sales orders. An outer join displays all customers, even those who do not have orders.

QAD
Our Passion. Your Advantage.

Fields Container

The screenshot shows the 'View Maintenance' window with the following fields and annotations:

- Name:** inv001. Annotation: **Leave Table name blank** (with an arrow pointing to the Name field).
- Table:** (empty). Annotation: **Name the local variable "local-var01"** (with an arrow pointing to the Table field).
- Field Name:** local-var01. Annotation: **Name the local variable "local-var01"** (with an arrow pointing to the Field Name field).
- View Field Data:**
 - Label Term:** Qty Unallocated. Annotation: **Enter the label, format and data-type** (with an arrow pointing to the Label Term field).
 - Enable QXtend Outbound:** (checkbox, unchecked).
 - Format:** >>>>>>>>9.99<<<<. Annotation: **Enter the label, format and data-type** (with an arrow pointing to the Format field).
 - Data Type:** decimal. Annotation: **Enter the label, format and data-type** (with an arrow pointing to the Data Type field).
- Expression:** Id_qty_oh - Id_qty_all. Annotation: **The expression that defines the variable Id_qty_oh - Id_qty_all is here** (with an arrow pointing to the Expression field).

Your view is complete and can now be used when creating a browse

Buttons: Delete, Back, Next

View Maintenance (36.20.18)

QAD Proprietary BPC-12-BRO-300

In Character, access this screen by pressing Esc-M in the View Maintenance screen. This accesses the menus. In the Window menu, select Tables.

Filtering and Sorting Browses



Filtering and Sorting Browses

- ▲ Power browses have a Browse Options selection screen for accessing filter and sort options
- ▲ Filters limit the viewable records according to your specification

QAD Proprietary

BPC-12-BRO-311

These operations are initiated within the browse once it is running.

Filter Screen

Item Browse X

Excel Print Add to Favorites Chart Designer New Edit

Search (1)

Unit of Measure equals EA Search Clear All

Viewing 1 - 50 of 50 Records per page: 100

Item Number	Description	Description 2	Unit of Measure	Prod Line	Item Type	Promotion Group
486-66	486-66 Personal Computer		EA	8000		
05-0005	BARREL		EA	6000	FAB	
30-1000	BATTERIES, LONG-LIFE	LITHIUM 10-YEAR	EA	2000	ELEC	ELECTRON AC
30-100	BATTERIES,ALKALINE,STD.	ENDURACELL 1 YEAR	EA	2000	ELEC	ELECTRON AC
30-10000	BATTERIES HEAVY DUTY	LONG LIFE 5-YR	EA	2000	COMP	ELECTRON AC
88-300	BOTTLE,PLASTIC,1 LITER		EA	3000	COMP	PROCESS AC
99-500	BOX,SHIPPER		EA	6000	COMP	
99-510	BOX,SHIPPER,LARGE		EA	1500	COMP	
88-100	CASE, AUTO-UNIT		EA	1500	COMP	AUTO AC
88-200	CASE,HOME/INDUST UNIT		EA	6000	COMP	DISCRETE AC
44-2000	COMPUTER CONTROL UNIT	MAIN-BOARD	EA	2000	COMP	ELECTRON AC
02-0001	CONFIGURED DISPLAY RACK		EA	1000	CONFIG	CARDED KL PG4
44-5000	CONTROL PANEL, LCD, STD.	44-SERIES ITEMS	EA	2000	COMP	ELECTRON AC
44-110	CONTROL UNIT, AUTOMOTIVE		EA	1500	COMP	ELECTRON AC
44-100	CONTROL UNIT, HOME USE		EA	2000	COMP	ELECTRON AC
22-110	CORD,POWER,EUROPE,WEST		EA	8000	COMP	DISCRETE AC
22-120	CORD,POWER,UK		EA	8000	COMP	DISCRETE AC
22-130	CORD,POWER,UNIVERSAL	ONE SIZE FITS ALL	EA	8000	COMP	DISCRETE AC
22-100	CORD,POWER,USA		EA	8000	COMP	DISCRETE AC

Only display items that have a UM of "EA"

Choose to sort by description

Item Browse (1.6)

Choosing Columns for Browse

Item Browse

Excel Print Add to Favorites Chart Chart Designer New Edit

Search (1)

Unit of Measure equals EA Search Clear All

Viewing 1 - 50 of 50 Records per page: 100

Item Number	Description	Description 2	Unit of Measure	Prod Li	Item Type	Group	Status	Buyer/Planner	Design Group	Promotion Group
22-100	CORD,POWER,USA		EA	8000	COMP	DISCRETE	AC			
22-110	CORD,POWER,EUROPE,WES		EA	8000	COMP	DISCRETE	AC			
22-120	CORD,POWER,UK		EA	8000	COMP	DISCRETE	AC			
22-130	CORD,POWER,UNIVERSAL	ONE SIZE FITS ALL	EA	8000	COMP	DISCRETE	AC			
486-66	486-66 Personal Computer		EA	8000		ELEC-PC				
05-0005	BARREL		EA	6000	FAB	RAW		JJ		
05-0075	LEAD CARTRIDGE		EA	6000	COMPNT	PUR		PH		
05-0085	END WITH CLIP		EA	6000	COMPNT	PUR		PH		
09-0001	PACKAGING BLISTER SEAL		EA	6000	COMPNT	PUR		KT		
				6000	COMPNT	PUR		KT		
				6000	COMPNT	PUR		JJ		
				6000	COMPNT	PUR		JJ		
				6000	COMPNT	PUR		LD		
55-100	EVAPORATOR, SERIES 10K		EA	6000	COMP	DISCRETE	AC			
66-100	MOTOR,440AC3PHASE		EA	6000	COMP	DISCRETE	AC			
66-110	MOTOR,7500WATT 110/22AC		EA	6000	COMP	DISCRETE	AC			
77-100	PUMP,1000GPH,SERIES 10K		EA	6000	COMP	DISCRETE	AC			
77-110	PUMP,1000GPH,SERIES 10K		EA	6000	COMP	DISCRETE	AC			
77-120	PUMP,1000GPH SERIES 10K		EA	6000	COMP	DISCRETE	AC			

**Click on the column you want to display,
Click again to reverse sort**

Item Browse (1.6)

Item Browse

Item Browse X

Excel Print Add to Favorites Chart Chart Designer New Edit

Search (1)

Unit of Measure equals EA Search Clear All

Viewing 1 - 50 of 50 Records per page: 100

Item Number	Description	Description 2	Unit of Measure	Prod Li	Item Type	Group	Status	Buyer/Planner	Design Group	Promotion Group
22-100	CORD,POWER,USA		EA	8000	COMP	DISCRETE	AC			
22-110	CORD,POWER,EUROPE,WEST		EA	8000	COMP	DISCRETE	AC			
22-120	CORD,POWER,UK		EA	8000	COMP	DISCRETE	AC			
22-130	CORD,POWER,UNIVERSAL	ONE SIZE FITS ALL	EA	8000	COMP	DISCRETE	AC			
486-66			EA	8000		ELEC-PC				
05-0075			EA	6000	FAB	RAW		JU		
05-0085	END WITH CLIP		EA	6000	COMPNT	PUR		PH		
09-0001	PACKAGING BLISTER SEAL		EA	6000	COMPNT	PUR		KT		
09-0035	PRINTED CARD - PENCIL		EA	6000	COMPNT	PUR		KT		
			EA	6000	COMPNT	PUR		JU		
			EA	6000	COMPNT	PUR		JU		
			EA	6000	COMPNT	PUR		LD		
66-100	MOTOR,440AC3PHASE		EA	6000	COMP	DISCRETE	AC			
66-110	MOTOR,7500WATT 110/22AC		EA	6000	COMP	DISCRETE	AC			
77-100	PUMP,1000GPH,SERIES 10K		EA	6000	COMP	DISCRETE	AC			
77-110	PUMP,1000GPH,SERIES 10K		EA	6000	COMP	DISCRETE	AC			
77-120	PUMP,100K.GPH SERIES 10K		EA	6000	COMP	DISCRETE	AC			

Search for a specific item

This is a link to other documents. Click and it jumps to Item Master Maintenance

Item Browse (1.6)

QAD
Our Passion. Your Advantage.

Output Setup

Choose Print to change the output settings, or use the direct link to Excel

Item Number	Description	Description 2	Unit of Measure	Prod Li	Item Type	Group	Status	Buyer/Planner	Design Group	Promotion Gro
22-100	CORD,POWER,USA		EA	8000	COMP	DISCRETE	AC			
22-110	CORD,POWER,EUROPE,WEST		EA	8000	COMP	DISCRETE	AC			
22-120	CORD,POWER,UK		EA	8000	COMP	DISCRETE	AC			
22-130	CORD,PO					DISCRETE	AC			
486-66	486-66 Ph					FILE-PC				
05-0005	BARREL					RAW		JJ		
05-0075	LEAD CA					PUR		PH		
05-0085	END WITH CLIP		EA	6000	COMPNT	PUR		PH		
09-0001	PACKAGING BLISTER SEAL		EA	6000	COMPNT	PUR		KT		
09-0035	PRINTED CARD - PENCIL		EA	6000	COMPNT	PUR		KT		
12-4010	CUT PLATES - 12 x 15	STAINLESS STEEL	EA	6000	COMPNT	PUR		JJ		
12-4020	STEEL SHAFT FOR DISPLAY		EA	6000	COMPNT	PUR		JJ		
12-4100	WELDING RODS		EA	6000	COMPNT	PUR		LD		
55-100	EVAPORATOR, SERIES 10K		EA	6000	COMP	DISCRETE	AC			
66-100	MOTOR,440AC3PHASE		EA	6000	COMP	DISCRETE	AC			
66-110	MOTOR,7500WATT 110/22AC		EA	6000	COMP	DISCRETE	AC			
77-100	PUMP,1000GPH,SERIES 10K		EA	6000	COMP	DISCRETE	AC			
77-110	PUMP,1000GPH,SERIES 10K		EA	6000	COMP	DISCRETE	AC			
77-120	PUMP,100K.GPH SERIES 10K		EA	6000	COMP	DISCRETE	AC			

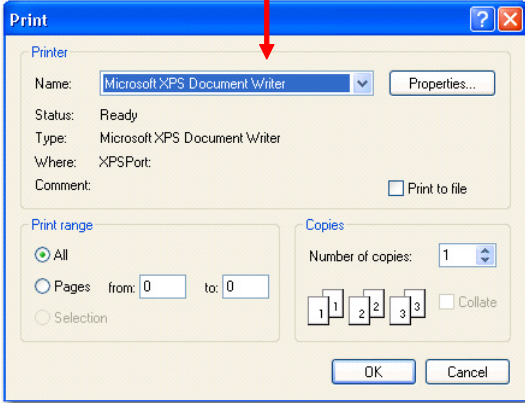
QAD Proprietary

BPC-12-BRO-350

Print Browses



Print options

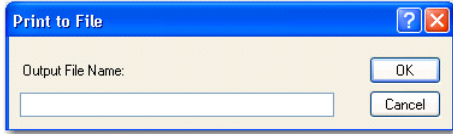


The 'Print' dialog box shows the printer 'Microsoft XPS Document Writer' selected. It includes options for 'Print range' (All, Pages, Selection) and 'Copies' (Number of copies, Collate). A red arrow points from the 'Print to file' checkbox to the explanatory text box.

This Dialog box will come up if you did not choose to send it to file

OR

Choose the directory to put the file in



The 'Print to File' dialog box prompts for an 'Output File Name' and has 'OK' and 'Cancel' buttons. A red arrow points from the text box to this dialog.

QAD Proprietary

BPC-12-BRO-376



Print Results

ppbr013.p		1.6 Item Browse		Date: 04/28/03	
Page: 1		QAD DB 05		Time: 15:51:40	
Item Number	Description	Unit of Measure	Product Line	Status	
02-0009	Packaged Pen	EA	6000	AC	
0300000	Unit, Transaxle Aux	EA	1000	ac	
0300001	Axle, Integral Arm Steel	EA	1000	ac	
0300002	Filter, Air	EA	1000	ac	
0300003	Filter, Coolant	EA	1000	ac	
04-0009	Asseembled Pen	EA	6000	AC	
05-005	Barrel	EA	6000	AC	
05-0079	Ink Cart. 50 ML	EA	6000	AC	
05-0095	Pen Cap	EA	6000	AC	
05-0579	Prototype Ink Cart.HQ	EA	6000	AC	
09-0039	Printed Card Pen	EA	6000	AC	
0924218-0000	w/wire .045CB 40#	LB	1500	AC	
0924743-0002	w/wire .035 409CB 40#	LB	1500	AC	
0990689-0002	U-Bolt Clamp	EA	1500	AC	
0990711-0007	U-Bolt Clip	EA	1500	AC	
0991150-0008	Rod Bracket	EA	1500	AC	
0992913-0009	Bracket	EA	1500	AC	
0993792-0008	Bracket	EA	1500	AC	
0993793-0003	Bracket	EA	1500	AC	
0994485-0008	MLF-SHL/WRAP 7.24x11.02	EA	1500	AC	
0995576-0018	M & E Assy 7.24 x 11.02	EA	1000	AC	
1-120/119	Ring, Retaining Internal	EA	5000	AC	

Browse Graphing


Our Passion. Your Advantage.

Browse Graphing

Item Browse
Excel Print Add to Favorites Chart **Chart Designer** New Edit

Search (1)

Unit of Measure equals EA Search Clear All

Viewing 1 - 50 of 50 Records per page: 100

< < > >

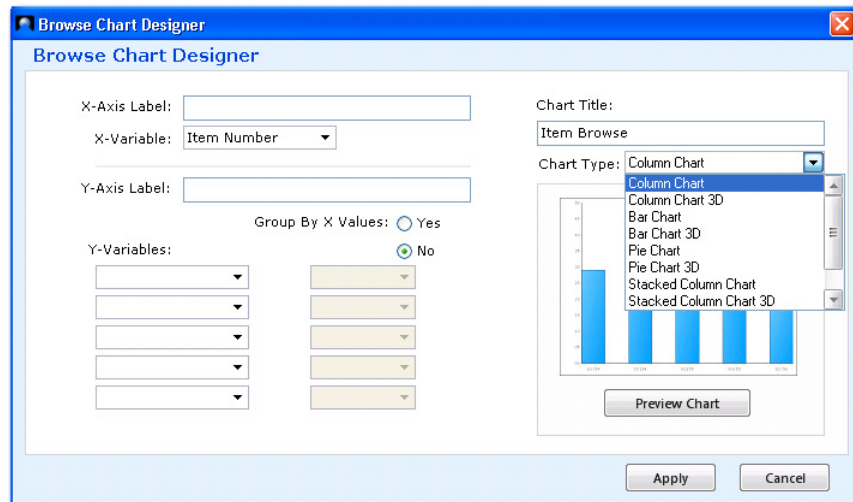
Description	Description 2	Unit of Meas	Prod L	Item Num	Item	Group	Sta	Buyer/Plan	Design Gr	Promotion Gr	Drawn
MECHANICAL PENCIL	(5MM) BLISTER PACKED	EA	1000	02-0005	FINGOOD	CARDED		PL		PG5	
ERASER REFILL PACK		EA	1000	02-0010	FINGOOD	CARDED		PL		PG6	
LEAD REFILL PACK		EA	1000	02-0020	FINGOOD	CARDED		PL		PG6	
DISPLAY RACK		EA	5000	03-0030	FINGOOD	CARDED		PH		PG4	
NOMAD(TM) COOLING SYSTEM	AUTOMOTIVE UNIT	EA	1500	10-15000	FINGOOD	AUTO	AC				10-15000
KOOLAIRE(TM) COOLANT	STANDARD, SCENTED	EA	3000	90-100	FINGOOD	PROCESS	AC				90-100
KOOLAIRE(TM) MEDIC-GRADE	COOLER CONDITIONER	EA	3000	90-110	FINGOOD	PROCESS	AC				90-110
PLANNING ITEM	PENCIL Family	EA						KL			
PENCIL ASSEMBLY		EA						PH			702
BARREL		EA						JJ			
BATTERIES ALKALINE STD.	ENDURACELL 1 YEAR	EA	2000	30-100	ELEC	ELECTRON	AC				30-100
BATTERIES, LONG-LIFE	LITHIUM 10-YEAR	EA	2000	30-1000	ELEC	ELECTRON	AC				
CONFIGURED DISPLAY RACK		EA	1000	02-0001	CONFIG	CARDED		KL		PG4	
OASIS(TM) COOLING SYSTEM	HOME/INDUST MODEL	EA	1500	10-10000	CONFIG	DISCRETE	AC				10-10000
LEAD CARTRIDGE		EA	6000	05-0075	COMPNT	PUR		PH			792
END WITH CLIP		EA	6000	05-0085	COMPNT	PUR		PH			
PACKAGING BLISTER SEAL		EA	6000	09-0001	COMPNT	PUR		KT			
PRINTED CARD - PENCIL		EA	6000	09-0035	COMPNT	PUR		KT			
CUT PLATES - 12 x 15	STAINLESS STEEL	EA	6000	12-4010	COMPNT	PUR		JJ			
STEEL SHAFT FOR DISPLAY		EA	6000	12-4020	COMPNT	PUR		JJ			
WELDING RODS		EA	6000	12-4100	COMPNT	PUR		LD			

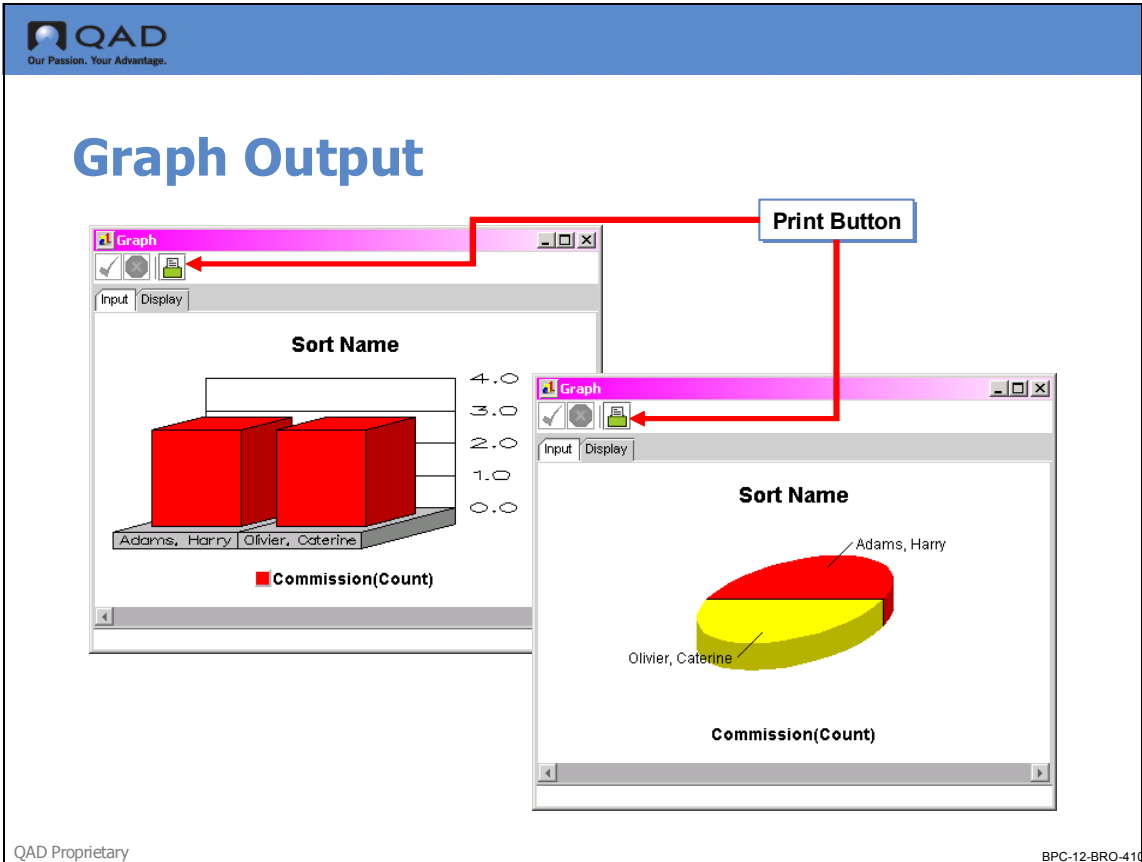
Chart, and Chart Designer

Item Browse (1.6)

QAD Proprietary
BPC-12-BRO-399

Graph Selection



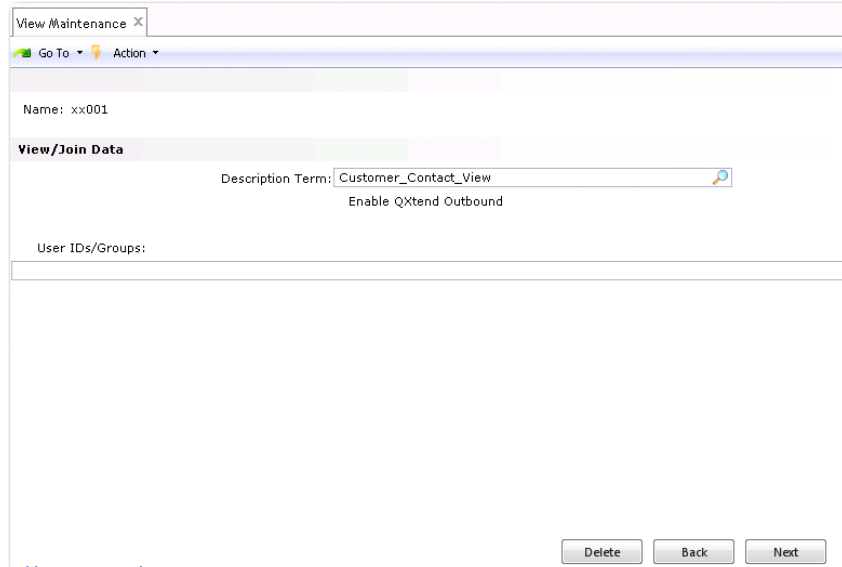


Hands On Exercise: Browse Lab

- Create a browse that displays the customer contact information, including our new e-mail information
 - Include customer number, name, and the contact information

Solution: Create a View

- ▲ Need to create a view of the ad_mstr and the clad_mstr tables



View Maintenance

Go To Action

Name: xx001

View/Join Data

Description Term: Customer_Contact_View

Enable QXtend Outbound

User IDs/Groups:

Delete Back Next



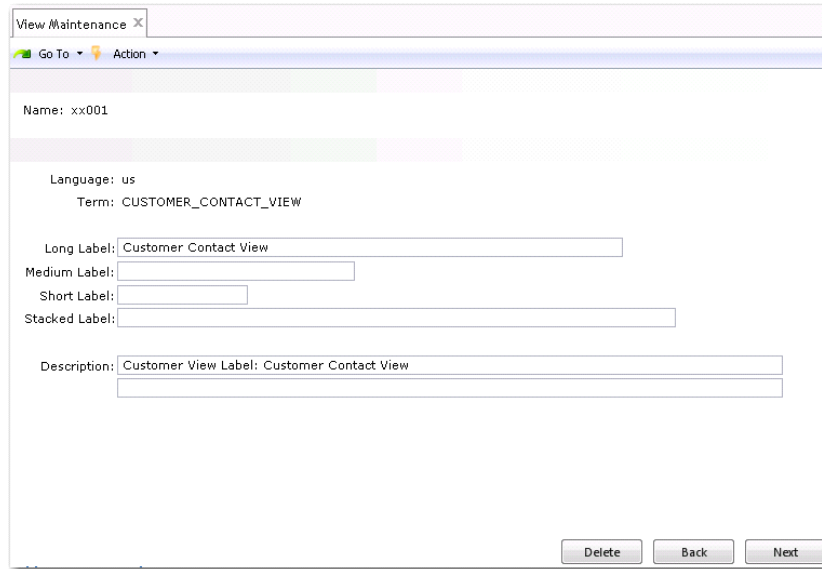
Solution: Add Labels

- ▲ Need to create Label Term if it does not exist

The screenshot shows a 'View Maintenance' dialog box with a title bar and a close button. Below the title bar is a 'Go To' field and an 'Action' dropdown menu. The 'Name' field contains 'xx001'. The 'View/Join Data' section shows 'Description Term: CUSTOMER_CONTACT_VIEW' and 'Enable QXtend Outbound'. Below this is a 'User IDs/Groups' field. A warning message box is displayed in the center, stating 'WARNING: Label term does not exist. Create.' with 'Yes' and 'No' buttons. At the bottom right are 'Back' and 'Next' buttons.

Solution: Add Label

- ▲ Add the Actual Label information and a description



The screenshot shows a 'View Maintenance' dialog box with a title bar and a close button. Below the title bar is a toolbar with 'Go To' and 'Action' buttons. The main area contains the following fields:

- Name: xx001
- Language: us
- Term: CUSTOMER_CONTACT_VIEW
- Long Label: Customer Contact View
- Medium Label:
- Short Label:
- Stacked Label:
- Description: Customer View Label: Customer Contact View

At the bottom right, there are three buttons: Delete, Back, and Next.



Solution: Create a View

- ▲ The first table in our join is ad_mstr

View Maintenance X

Go To Action

Name: xx001

Sequence: 1

Table Selection

Table: ad_mstr Join Type: Inner

Join Phrase:

Delete Back Next

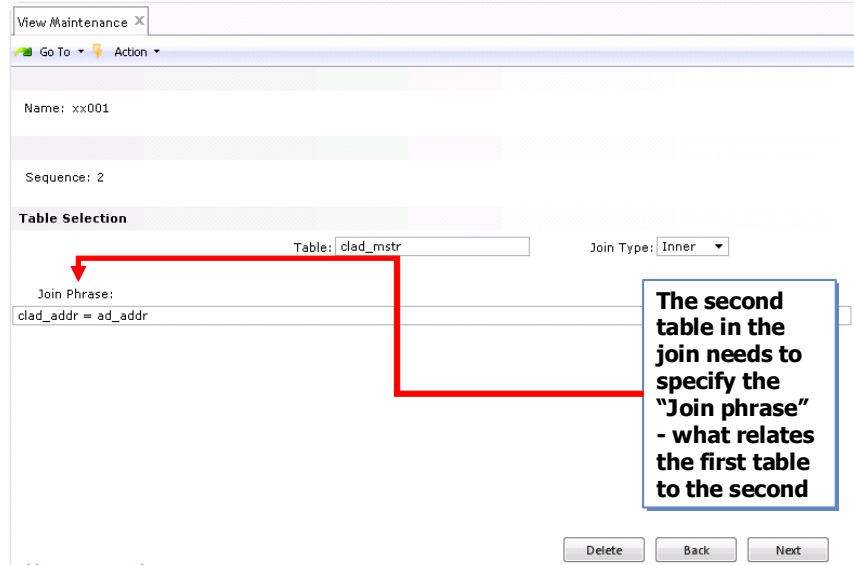
ad_mstr is the first table in the join; note the sequence" number

QAD Proprietary

BPC-12-BRO-470

Solution: Create a View

- ▲ The second table in the join is c1ad_mstr



The screenshot shows the 'View Maintenance' window with the following details:

- Name: xx001
- Sequence: 2
- Table Selection**
 - Table: clad_mstr
 - Join Type: Inner
 - Join Phrase: clad_addr = ad_addr

A red arrow points from the 'Table: clad_mstr' field to the 'Join Phrase' field. A blue callout box on the right contains the text: 'The second table in the join needs to specify the "Join phrase" - what relates the first table to the second'.

Buttons at the bottom: Delete, Back, Next.

Solution: Creating the Browse

- ▲ Select a name for the browse; remember the naming conventions

Provide a detailed description of what the browse will display

Select the browses types.

In this browse, the "view" is the view we just created. Remember if you are not browsing a created "view" then the view would be a table name.

QAD Desktop 2.1.1.1 (3) [UserName] - Microsoft Internet Explorer

Logout | Support | Help | MEGx.net

Browse Maintenance

Name: xx001 < Preview >

Browse Data

Description Term: Customer Contact Browse

View: xx001

Power Browse: Yes

Look-up Browse: Yes

Where:

F1=Go 2=Help 3=Ins 4=End 5=Delete 7=Recall 8=Clear

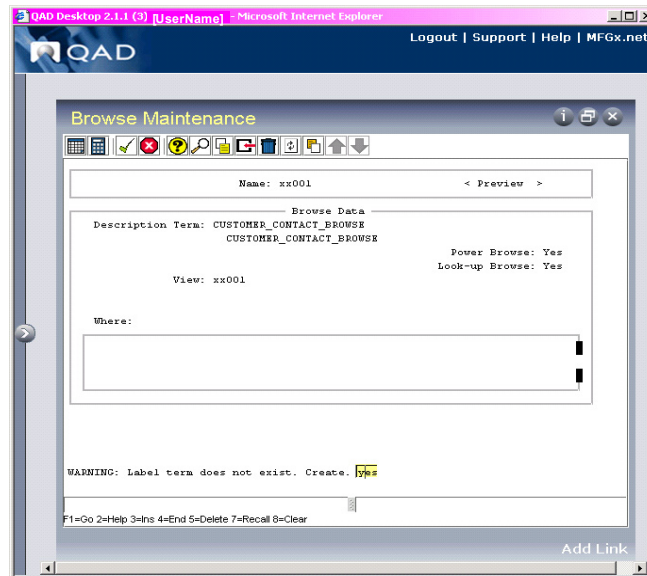
Add Link

QAD Proprietary

BPC-12-BRO-490

Solution: Add Labels

- ▲ Need to create Label Term if it does not exist



QAD Desktop 2.1.1.1 [UserName] - Microsoft Internet Explorer

Logout | Support | Help | MFGx.net

Browse Maintenance

Name: xx001 < Preview >

Browse Data

Description Term: CUSTOMER_CONTACT_BROWSE
CUSTOMER_CONTACT_BROWSE

View: xx001

Power Browse: Yes
Look-up Browse: Yes

Where:

WARNING: Label term does not exist. Create. **Yes**

F1=Go 2=Help 3=Ins 4=End 5=Delete 7=Recall 8=Clear

Add Link



Solution: Add Label

- ▲ Add the Actual Label information and a description

QAD Desktop 2.1.1 (3) [UserName] Microsoft Internet Explorer

Logout | Support | Help | MFGx.net

Browse Maintenance

Name: xx001 < Preview >

Language: US
Term: CUSTOMER_CONTACT_BROWSE

Long Label: Customer Contact Browse
Medium Label:
Short Label:
Stacked Label:
Description:

Adding new record.
F1=Go 2=Help 3=Ins 4=End 5=Delete 7=Recall 8=Clear

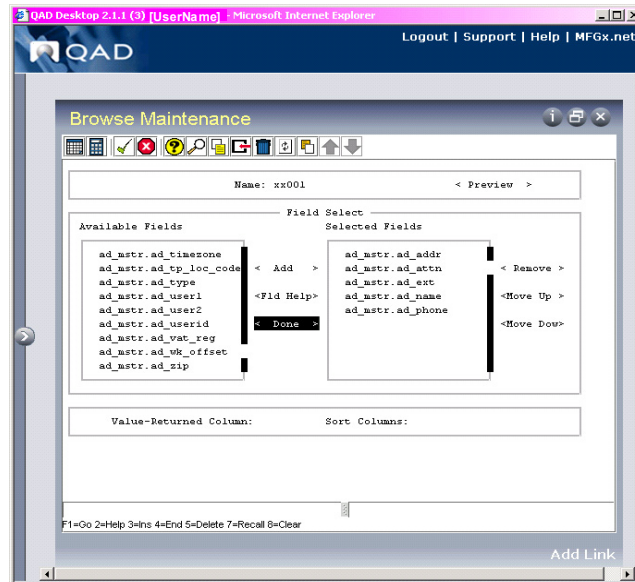
Add Link

QAD Proprietary

BPC-12-BRO-510

Solution: Creating the Browse

- ▲ Select the fields for the browse display

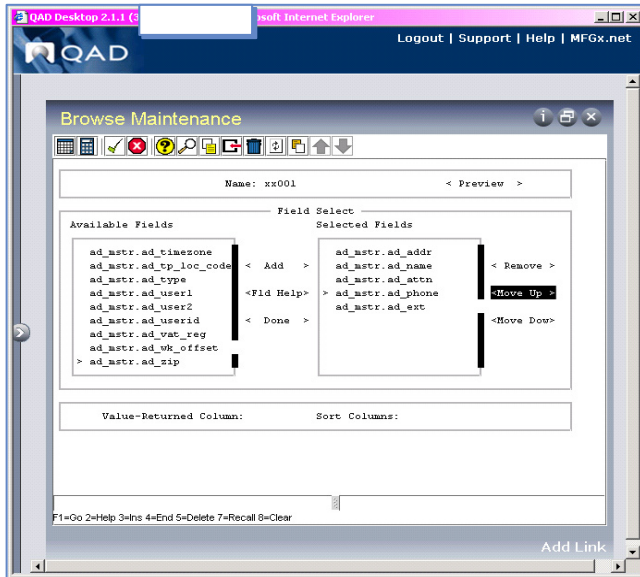


Select fields by scrolling through the fields and select by pressing the spacebar

Press Tab to get to the Add button, then press Enter to move the fields to the Selected Fields" box

Solution: Creating the Browse

- ▲ Move the fields into a logical order



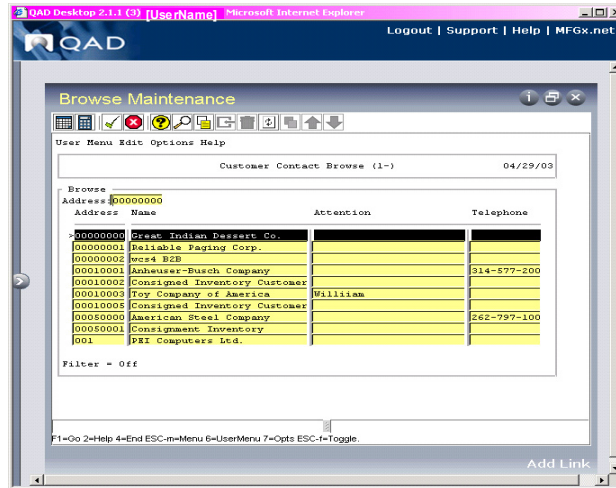
Select the field and tab to the "Move Up" or "Move Down" buttons to position the fields in the proper order

QAD Proprietary

BPC-12-BRO-530

Solution: Creating the Browse

- ▲ Go through the rest of the screens until you get back to the first screen.
 - Press Enter to preview the Browse!



QAD Desktop 2.1.1 (9) [UserName] Microsoft Internet Explorer

Logout | Support | Help | MFGx.net

Browse Maintenance

User Menu Edit Options Help

Customer Contact Browse (1-) 04/29/03

Browse

Address 00000000

Address	Name	Attention	Telephone
00000000	Great Indian Dessert Co.		
00000001	Reliable Paging Corp.		
00000002	West ESB		
00010001	Anheuser-Busch Company		314-577-200
00010002	Consigned Inventory Customer		
00010003	Toy Company of America	William	
00010005	Consigned Inventory Customer		
00050000	American Steel Company		262-797-100
00050001	Consignment Inventory		
001	P&I Computers Ltd.		

Filter = Off

F1=Go 2=Help 4=End ESC=m=Menu 6=UserMenu 7=Opts ESC=t=Toggle

Add Link



Solution: What else?

- ▲ Move generated code to custom source directory
- ▲ Compile browse source
- ▲ Assign a Menu number?

Review



Browses

- ✓ Look-up browse
- ✓ Power browse
- ✓ Views
- ✓ Tables
- ✓ Designing browses
- ✓ Printing browses
- ✓ Graph browses

Chapter 13

Customizing On-Line Help

Introduction



Customizing On-Line Help

In this section, you learn about:

- ▲ Two help formats
 - mfghelp.db
 - Winhelp

Customizing QAD Enterprise Applications Help



Customizing QAD Enterprise Applications Help

- ▲ Two types of Help Formats
 - Records in the QAD Enterprise Applications help database
 - Winhelp files
- ▲ mfghelp.db
 - Database created at install time
 - Data is language specific
 - fieldhelp.fhd file is loaded thru an QAD Enterprise Applications session in Field Help Load
 - Available for character or GUI clients
- ▲ Winhelp
 - Shipped as run-time .hlp files
 - Available for GUI clients only



Customizing Field and Procedure Help

1. In User Maintenance, verify that windows help is inactive
 2. Determine the field name (Ctl^F) and the procedure name by clicking help in the menu bar, then click about
 3. Go to Field Help Maintenance
 - Calling Field: This is the Field name from step 2
(To add Procedure Help, leave Calling Field blank.)
 - Calling Proc: This is the procedure name from step 2
 - Enter the Help Text.
- ▲ Note: The text entered in Field Help Maintenance appears before the field and procedure help supplied by QAD.



QAD Enterprise Applications WINHELP - GUI

- ▲ .hlp files are found in the <LangDir>/help directory
 - us/help/so.hlp - procedure help for all so*.p programs
 - us/help/aflds.hlp - field help for all fields beginning with the letter "a"
- ▲ .hlp files are run-time files - not readable
- ▲ Manually created for each release
 - Process not supported for customer use
- ▲ Customers should toggle WINHELP off in User Maintenance and use customized field and procedure help from the mfghelp database

How Winhelp Files Get Created

- ▲ Start with a Word (.doc) file
 - Not shipped with QAD Enterprise Applications release
- ▲ Apply style tags for fields, programs, xref
 - Defined in the Word file
- ▲ Attach a help template to the word files (.rtf's)
- ▲ Apply Winhelp Standards for footnotes, headings and jumps to the .rtf
- ▲ Create Winhelp project file (.hpj)
- ▲ Use Microsoft Help Compiler to create .hlp files
- ▲ Move to <LangDir>/help

Review



Customizing On-Line Help

- ✓ Two help formats
 - mfghelp.db
 - Winhelp

Chapter 14

CIM Interface

Introduction



CIM Interface

In this section, you learn about:

- ▲ CIM Mechanism
- ▲ Rationale for using CIM
- ▲ CIM Data File
- ▲ CIM Test Program
- ▲ Loading CIM Data Files
- ▲ Processing CIM Data Files

CIM Mechanism



CIM Mechanism

- ▲ PROGRESS allows redirecting input from
 - File
 - Process (UNIX)
- ▲ QAD has written programs to load and process formatted ASCII files
- ▲ CIM allows managing and reporting on what's been loaded

QAD Proprietary

BPC-14-CIM-024

PROGRESS allows redirecting the input stream from the keyboard to a file (or a process in Unix).

Rationale for Using CIM



Rationale for Using CIM

- ▲ CIM processing allows the user to load data using input files
- ▲ CIM can also use input streams such as a bar code reader or other data collection devices
- ▲ CIM is often used by new QAD Enterprise Applications customers to load data from other systems
- ▲ CIM can be used to build interfaces with other applications

QAD Proprietary

BPC-14-CIM-039

- CIM allows great flexibility because data can be loaded from ASCII files created by other software packages. This is especially useful when you convert from another software package to QAD Enterprise Applications. Use this functionality to load shop floor data, interface with other enterprise applications, and so on.
- QAD Enterprise Applications allows ASCII data to be loaded in three different ways:
 - Loading tables/files that have been dumped (either through the Data Dictionary utilities or programmatically)
 - Reloading files that have been archived with QAD Enterprise Applications delete and archive programs
 - Loading files through the CIM interface
- The CIM interface takes specially formatted output from other databases or from devices like bar-code readers and loads the output into the system. The advantage of using CIM to load your data (as opposed to directly loading data into a table like the customer master) is that CIM uses the actual QAD Enterprise Applications programs to load the data, which means that the programs check the data as it is loaded. Since QAD Enterprise Applications uses programmatic enforcing of the linkages between tables and maintains data in tables that may not be inherently obvious, this is of critical importance. While this is very important from a data integrity perspective, it's not as fast as just importing a table from an ASCII text file.

CIM Data Formatting



CIM Data Formatting

- ▲ Single line of data for each input request
- ▲ Use ~ for continuation of lines
- ▲ Fields are separated by spaces
- ▲ Character fields surrounded by quotes (" ")
- ▲ Dates formatted as 99/99/99
- ▲ Use a hyphen (-) to tab over fields
- ▲ End of input line is the same as the GO key
- ▲ A single period (.) is processed as END key
- ▲ In DOS files, there should be no (CTRL Z) EOF embedded in the file

QAD Proprietary

BPC-14-CIM-04

These are the basic rules that apply to formatting data so that the CIM processor (and PROGRESS) can understand what you want it to do.

There is no way to simulate an F5 key, therefore deletes are not possible through CIM.

CIM File Format


Our Passion. Your Advantage.

CIM File Format

```

@@batchload <program-name.p>

[input-data ...]

@@end

@@batchload <program-name.p>

[input-data ...]

@@end

...

```

QAD Proprietary
BPC-14-CIM-05

Each QAD Enterprise Applications program processes data in groups. Within a frame, a group consists of the fields that are highlighted at any one time. Data going into the CIM load must be delimited by groups. For each group, @@batchload signals the beginning of the group, and @@end indicates the end of the group. The program-name is the name of the QAD Enterprise Applications program that receives the data. The input data must have the same structure as if it were entered from the terminal.

Multiple transactions can be handled between the @@batchload and @@end statements. However, for error processing, it is better to keep each transaction separate. That way, if an error occurs, only the errored transaction is affected rather than the whole group. The CIM processor expects to see a header and a footer on each data record. This lets the processor know that it is the correct type of file and also the name of the program that the data will be loaded into.

Building CIM Files



Building CIM Files...

- ▲ Manually...
- ▲ Programmatically

QAD Proprietary

BPC-14-CIM-06

There is really not much sense in building these files by hand, one at a time except the first time, when you are figuring out how the file should be structured. The real benefit comes from developing a program that will take data from somewhere, often another software application, and format it so that it can be loaded into QAD Enterprise Applications with CIM. This may be a one-time occurrence, during your implementation and cut over from an old system, or you may have an ongoing need for continuing interfacing that CIM will serve your needs.



Figuring Out Format

- ▲ Open two windows
 - QAD Enterprise Applications
 - Text editor
- ▲ Step through the QAD Enterprise Applications program by field
- ▲ Record data/format in text editor
- ▲ Test
- ▲ Develop program to generate the formatted file(s)

Customer Maintenance

Customer Maintenance X

Go To Action

Customer Address

Customer: mot-002

Name:

Address:

Address:

Address:

City: State: Post: Format: 0

Country: County:

Attention: [2]:

Telephone: Ext: [2]: Ext:

Fax/Telex: [2]: Added: 11/20/2008

Customer Data

Sort Name: Type:

Salespsn1: Multiple: ☒ Region:

Ship Via: Currency: USD

AR Acct: 1200 Dual Pricing Cur:

Resale: Site:

Remarks: Lang:

[Adding new record](#)

Delete Back Next

CIM Data File



Our Passion. Your Advantage.

CIM Data File

▲ "mot-002"

QAD Proprietary

BPC-14-CIM-099

Customer Maintenance

Customer Maintenance X

Go To Action

Customer Address

Customer: mot-002

Name: Motorola

Address: 1303 East Algonquin Road

Address:

Address:

City: Schaumburg State: IL Post: 60196 Format: 0

Country:

County:

Attention: Robert Galvin [2]:

Telephone: 555-867-1000 Ext: [2]: Ext:

Fax/Telex: [2]: Added: 11/20/2008

Customer Data

Sort Name:

Salespsn1: Multiple: ☒

Ship Via:

AR Acct: 1200

Resale:

Remarks:

Type:

Region:

Currency: USD

Dual Pricing Cur:

Site:

Lang:

Delete Back Next

Adding new record



CIM Data File

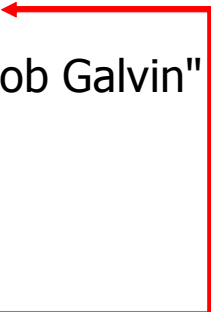
"mot-002"

"Motorola" "1303 East Algonquin Road" - -

"Schaumburg" "IL" ~

"60196" "0" "USA" - "Bob Galvin" "555-867-5309"

"0000"



Notice the "~" tilde for line continuation...

CIM example

Customer Maintenance X

Go To Action

Customer Address

Customer: mot-002
 Name: Motorola
 Address: 1303 East Algonquin Road
 Address:
 Address:
 City: Schaumburg State: IL Post: 60196 Format: 0
 Country: United States of America usa County:
 Attention: Robert Galvin [2]:
 Telephone: 555-867-1000 Ext: [2]: Ext:
 Fax/Telex: [2]: Added: 11/20/2008

Customer Data

Sort Name: Second Sort Name Type:
 Salespsn1: Multiple: ☒ Region:
 Ship Via: UPS Currency: USD
 AR Acct: 1200 Dual Pricing Cur:
 Resale: Site: 10000
 Remarks: Was customer #MOT Lang:
 Back Next



CIM Data File

"mot-002"

"Motorola" "1303 East Algonquin Road" - -
"Schaumburg" "IL" ~

"60196" "0" "USA" - "Bob Galvin" "555-867-5309"
"0000"

"Second Sort Name" - no "UPS" - - - - "was
customer # MOT" - - ~

- "10000" -

<Blank> => (Blank line for "GO" through
additional tax information frame.)


Our Passion. Your Advantage.

CIM example

Customer Maintenance X

Go To Action

Customer Address

Customer: mot-002

Name: Motorola

Address: 1303 East Algonquin Road

Address:

Address:

City: Schaumburg	State: IL	Post: 60196	Format: 0
Country: United States of America	usa	County:	
Attention: Robert Galvin	[2]:		
Telephone: 555-867-1000	Ext: [2]:	Ext:	
Fax/Telex:	[2]:	Added: 11/20/2008	

Customer Data

Taxable: ☐

Price Table:

Discount Tbl:

Fixed Price: ☒

Class:

SIC:

Partial OK: ☒

Invoice by Authorization: ☐

BackNext

QAD ProprietaryCustomer Maintenance (2.1.1)BPC-14-CIM-140



CIM Data File

"mot-002"

"Motorola" "1303 East Algonquin Road" - - "Schaumburg"
"IL" ~

"60196" "0" "USA" - "Bob Galvin" "555-867-5309" "0000"

"Second Sort Name" - no "UPS" - - - - "was customer #
MOT" - - ~

- "10000" -

<Blank> => (Blank line for "GO" through additional tax
information frame.)

Blank line for "GO"
through tax
information frame.

QAD
Our Passion. Your Advantage.

CIM Example

Customer Maintenance ✕

Go To
Action

Customer Address

Customer: mot-002

Name: Motorola

Address: 1303 East Alginquin Road

Address:

Address:

City: Schaumburg

Country: United States of America

Attention: Robert Galvin

Telephone: 555-867-1000

Fax/Telex:

Address Tax Data

Taxable: ☐

Tax Zone:

Tax Class:

Tax Usage:

Tax In: ☐

Tax ID - Federal:

Tax ID - State:

Tax ID - Misc 1:

Tax ID - Misc 2:

Tax ID - Misc 3:

In City: ☒

Format: 0

Ext:

Added: 11/20/2008

Customer Data

Taxable: ☐

Price Table:

Discount Tbl:

Fixed Price: ☒

Back Next

QAD Proprietary
Customer Maintenance (2.1.1)
BPC-14-CIM-160



CIM Data File

"mot-002"

"Motorola" "1303 East Algonquin Road" - - "Schaumburg"
"IL" ~

"60196" "0" "USA" - "Bob Galvin" "555-867-5309" "0000"

"Second Sort Name" - no "UPS" - - - - "was customer #
MOT" - - ~

- "10000" -

<Blank> => (Blank line for "GO" through
additional tax information frame.)

CIM Example

Customer Maintenance x

Go To Action

Customer Address

Customer: mot-002
Name: Motorola
Address: 1303 East Alginquin Road
Address:
Address:
City: Schaumburg State: IL Post: 60196 Format: 0
Country: United States of America usa County:
Attention: Robert Galvin [2]:
Telephone: 555-867-1000 Ext: [2]: Ext:
Fax/Telex: [2]: Added: 11/20/2008

Customer Data

Sort Name: Second Sort Name Type: 
Salespsn1: NONE  Multiple: ☐ Region: MW 
Ship Via: UPS  Currency: USD 
AR Acct: 1200   Dual Pricing Cur:
Resale:  Site: 10000 
Remarks: Was customer #MOT Lang: 

Back Next



CIM Data File

"mot-002"

"Motorola" "1303 East Algonquin Road" - - "Schaumburg"
"IL" ~

"60196" "0" "USA" - "Bob Galvin" "555-867-5309" "0000"

"Second Sort Name" - no "UPS" - - - - "was customer #
MOT" - - ~

- "10000" -

<Blank> => (Blank line for "GO" through
additional tax information frame.)

"MW"



Our Passion. Your Advantage.

CIM Example

Customer Maintenance

Go To
Action

Customer Address

Customer: mot-002
Name: Motorola
Address: 1303 East Algonquin Road
Address:
Address:
City: Schaumburg State: IL Post: 60196 Format: 0
Country: United States of America usa County:
Attention: Robert Galvin [2]:
Telephone: 555-867-1000 Ext: [2]: Ext:
Fax/Telex: [2]: Added: 11/20/2008

Customer Data

Taxable: ☐ Class:
Price Table: SIC:
Discount Tbl: Partial OK: ☒
Fixed Price: ☒ Invoice by Authorization: ☐

Back Next

QAD Proprietary
Customer Maintenance (2.1.1)
BPC-14-CIM-200

CIM Data File

"mot-002"

"Motorola" "1303 East Algonquin Road" - - "Schaumburg" "IL" ~

"60196" "0" "USA" - "Bob Galvin" "555-867-5309" "0000"

"Second Sort Name" - no "UPS" - - - - "was customer # MOT" - - ~

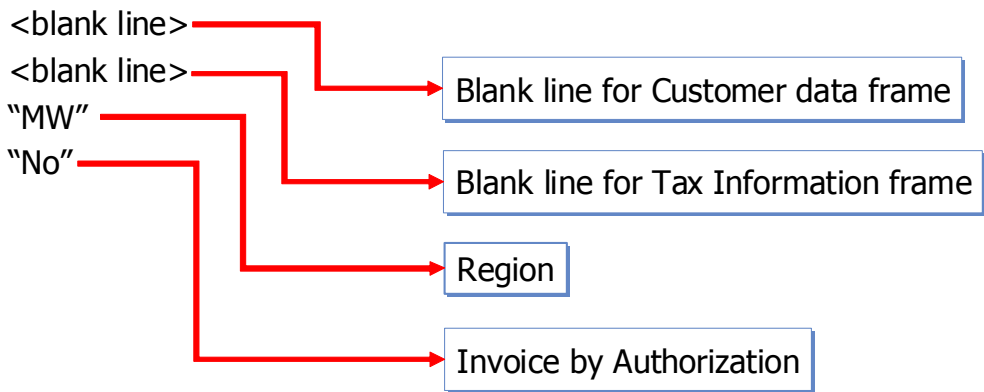
- "10000" -

<blank line>

<blank line>

"MW"

"No"



Blank line for Customer data frame

Blank line for Tax Information frame

Region

Invoice by Authorization

CIM Example

Customer Maintenance X

Go To Action

Customer Address

Customer: mot-002
 Name: Motorola
 Address: 1303 East Algonquin Road
 Address:
 City: Schaumburg State: IL Post: 60196 Format: 0
 Country: United States of America usa County:
 Attention: Robert Galvin [2]:
 Telephone: 555-867-1000 Ext: [2]: Ext:
 Fax/Telex: [2]: Added: 11/20/2008

Customer Credit Data

Credit Limit: 1000000 Disc Pct: 0.00 Bill To:
 Terms: Finance Charge: Last Credit Review:
 Credit Hold: Statement: Last Credit Update: 11/20/2008
 Credit Rating: Statement Cycle: High Credit: 0
 DB Number: Dun Letter: High Date:
 PO Required: Last Payment: Last Sale:

Back Next



CIM Data File

"mot-002"

"Motorola" "1303 East Algonquin Road" - - "Schaumburg" "IL" ~

"60196" "0" "USA" - "Bob Galvin" "555-867-5309" "0000"

"Second Sort Name" - no "UPS" - - - - "was customer # MOT" - - ~

- "10000" -

"MW"

no

1000000



Credit limit and end of line = "GO"

QAD
Our Passion. Your Advantage.

CIM Example

Customer Maintenance X

Go To
Action

Customer Address

Customer: mot-002
Name: Motorola
Address: 1303 East Algonquin Road
Address:
Address:
City: Schaumburg State: IL Post: 60196 Format: 0
Country: United States of America usa County:
Attention: Robert Galvin [2]:
Telephone: 555-867-1000 Ext: [2]: Ext:
Fax/Telex: [2]: Added: 11/20/2008

Customer Freight Data

Freight List: UPS-Red
Min Frt Wt: 5
Freight Terms: PrePaid

Back Next

QAD Proprietary Customer Maintenance (2.1.1) BPC-14-CIM-240

- Freight list must have an entry in Freight List Maintenance for the site and currency
- Freight terms must be entered using Freight Terms Maintenance



CIM Data File

```
"mot-002"
"Motorola" "1303 East Algonquin Road" - - "Schaumburg"
"IL" ~
"60196" "0" "USA" - "Bob Galvin" "555-867-5309" "0000"
"Second Sort Name" - no "UPS" - - - - "was customer #"
MOT" - - ~
- "10000" -
```

```
"MW"
no
1000000
"UPS-RED" 5 "prepaid"
```



Our Passion. Your Advantage.

CIM Example

Customer Maintenance X

Go To Action

Customer Address

Customer: mot-002

Name: Motorola

Address: 1303 East Algonquin Road

Address:

City: Schaumburg State: IL Post: 60196 Format: 0

Country: United States of America usa County:

Attention: Robert Galvin [2]:

Telephone: 555-867-1000 Ext: [2]: Ext:

Fax/Telex: [2]: Added: 11/20/2008

Enterprise Material Transfer Data

EMT Type: NON-EMT Customer Shipping LT: 0 Confirmed EMT SO: EMT Credit Flow:

Back Next

QAD Proprietary
Customer Maintenance (2.1.1)
BPC-14-CIM-260



CIM Data File

"mot-002"

"Motorola" "1303 East Algonquin Road" - - "Schaumburg" "IL" ~

"60196" "0" "USA" - "Bob Galvin" "555-867-5309" "0000"

"Second Sort Name" - no "UPS" - - - - "was customer # MOT" - - ~

- "10000" -

"MW"

No

1000000

"UPS-RED" 5 "prepaid"



Blank line to "GO" through EMT screen



Our Passion. Your Advantage.

CIM Example

Customer Maintenance X

Go To Action

Customer Address

Customer: mot-002
Name: Motorola
Address: 1303 East Algonquin Road
Address:
Address:
City: Schaumburg State: IL Post: 60196 Format: 0
Country: United States of America usa County:
Attention: Robert Galvin [2]:
Telephone: 555-867-1000 Ext: [2]: Ext:
Fax/Telex: [2]: Added: 11/20/2008

Bank Accounts

Bank	Acct Type	EDI	Branch	Bank Account	Begin Date	End Date

Back Next

QAD Proprietary
Customer Maintenance (2.1.1)
BPC-14-CIM-280

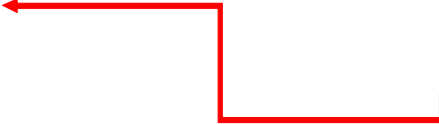


CIM Data File

```
"mot-002"  
"Motorola" "1303 East Algonquin Road" - - "Schaumburg" "IL" ~  
"60196" "0" "USA" - "Bob Galvin" "555-867-5309" "0000"  
"Second Sort Name" - no "UPS" - - - - "was customer # MOT" - - ~  
- "10000" -
```

```
"MV"  
no  
1000000  
"UPS-RED" 5 "prepaid"
```

▪



“.” for “END”, leaving this frame

CIM Testing



Key Pieces to CIM Testing

- ▲ An input file with your data
 - The text file we just created
- ▲ A program to step through CIM with your data
 - A test program that uses the text file as input to the QAD Enterprise Applications function
- ▲ The CIM processor
 - Once you verified your text file, test out the CIM processor

QAD Proprietary

BPC-14-CIM-300

The easiest way to create a CIM test file is to follow these steps:

- 1 Create two windows on your machine, one with the program you are going to test, the other with a text editor (it may or may not be the PROGRESS editor)
- 2 Step through the program on the one window, while recording your data in the editor window
- 3 Test the data with the test program used in this tutorial
- 4 Add the header and footer to the file
- 5 Test the program using the CIM processor

CIM Test Program



CIM Test Program

- ▲ Run in the PROGRESS Editor
 - Make sure your PROPATH is set correctly
 - Verify the data was input correctly, by running customer maintenance within QAD Enterprise Applications

```
/*CIM Test*/  
define shared variable batchrun as logical initial yes.  
input from <cim input file>.  
batchrun = yes.  
run us/ad/adcsmt.r.  
input close.
```

CIM Header and Footer

@@batchload adcsmt.p

CIM load header

"mot-006"

"Motorola" "1303 East Algonquin Road" - - "Schaumburg" "IL" ~

"60196" "0" "USA" - "Bob Galvin" "555-867-5309" "0000"

"Second Sort Name" - no "UPS" - - - - "was customer # MOT" - - ~

- "10000" -

"MW"

no

1000000

"UPS-RED" 5 "prepaid"

.

CIM load footer

@@end

Loading CIM Data



Loading CIM Data

CIM Data Load X

Go To Action

Input from File/Continuous Process: File

Input File/Continuous Process Name: cim.txt

Batch Data Load Groups Entered:
Group ID: To:

BackNext

QAD Proprietary

CIM Data Load (36.15.1)

BPC-14-CIM-33



Our Passion. Your Advantage.

Processing CIM Data Files

CIM Data Load Processor X

 Go To
  Action

Group ID: 0

To: 99999999

Repeat Processing: ☐

Pause Seconds Before Repeat: 300

Process Direct From File: ☐

Input File Name:

Output:

Back

Next

QAD Proprietary
 CIM Data Load Processor (36.15.2)
 BPC-14-CIM-340

The CIM Data Load Processor

mgbdp.p

**output to work_i_file
for each bdl_det:
 put bdl_data
output close.**

**input from work_i_file
output to work_o_file
run function_to_run
input close
output close.**

**input from work_o_file
parse ERROR, WARNING...
update bdl_pgm_errs bdl_pro_errs
input close.**



Our Passion. Your Advantage.

CIM Data Load Report

CIM Data Load Report/Delete X

 Go To
  Action

Group ID:
Date Entered:
Time Entered:
Date Processed:
Time Processed:

To:
To:
To:
To:
To:

Display Load Data: ☐
Report Processed Data with No Errors: ☐
Report Processed Data with Errors: ☒
Report Unprocessed Data: ☐
Delete Reported Data: ☐

Output:
Batch ID:

Back
Next

QAD Proprietary
CIM Data Load Report/Delete (36.15.3)
BPC-14-CIM-360



CIM Data Load Report

- ▲ Using CIM Data Load Report/Delete you can:
 - Generate a report on processed or unprocessed data load files
 - Selectively delete data load files.
- ▲ Setting Delete Reported Data to Yes deletes the specified data-load groups from the database
 - To delete only groups that have been processed successfully, you should also set Report Processed Data with No Errors to Yes.

CIM Data Load Report

- ▲ Warning: Setting Delete Reported Data to Yes and Report Processed Data with Errors or Report Unprocessed Data to Yes deletes data not yet loaded into the QAD Enterprise Applications data files.
- ▲ Deleted data cannot be recovered. You may want to back up your database before executing this function.
- ▲ You should define menu security for this program to restrict access

Why Test CIM?



Why Test CIM?

- ▲ Use “if not batchrun” to bracket these instances of Progress code:
 - Next-prompt with undo, retry have the effect of resetting the cursor to a previous frame field and attempting to process all subsequent CIM data starting with this field. As the CIM data cannot be repositioned dynamically to compensate, this action generates spurious and confusing CIM errors which can obscure the real error, delaying diagnosis and correction.
 - Pausing after displaying warnings/messages
 - Using dynamic PROGRESS frame variables
 - Any code using the CHOOSE command (for example, the swindow*.i include files)

There are some programs within QAD Enterprise Applications where CIM does not work. Generally, you should run CIM from a character client on the server. QAD’s method of developing the GUI front-end doesn’t always work with CIM.

Use of the “choose” statement in PROGRESS is not supported in CIM. In some places there are workarounds driven by the use of the “batchrun” variable, which unfortunately is not universally used.

CIM processing causes problems in certain frames such as the use of frames for costing elements in Item Cost Maintenance (program ppptmt03.p - subprogram ppptmta.p). The batchrun variable was used to skip over sections of code. As a result of using batchrun in your code, processing is skipped or is handled in a different way during batch and CIM processing.

If keyfunction is used in the program being executed through CIM, unexpected results may occur.

CIM Processing Drawbacks

- ▲ Database validations will not stop processing the data
- ▲ CIM processing is not very efficient
- ▲ CIM doesn't consistently work under GUI
- ▲ Progress does not permit the F5 key to be emulated by CIM, therefore CIM cannot be used to process standard QAD Enterprise Applications delete transactions

QAD Proprietary

BPC-14-CIM-400

- Database validations (like Generalized codes, Site Security, and so on) does not run when batchrun = yes, because .v files present in the PROGRESS data dictionary are not run in this mode. This can permit invalid data to update the QAD Enterprise Applications database.
- CIM does function in the NT environment when run in character mode pre-8.2 Progress versions, but only in file rather than continuous mode because Progress did not support pipes on NT until 8.2



CIM Processing Considerations

- ▲ CIM files should contain three Endkey emulations “.”, each occurring on separate lines) at the end of each CIM transaction
- ▲ The use of dynamic pop-up windows inside QAD Enterprise Applications cannot be accommodated by CIM when they are displayed only conditionally based on QAD Enterprise Applications data values, such as control file settings
- ▲ CIM can have slow performance due to the overhead of keystroke emulation and parsing as handled by Progress.
- ▲ It should be used for fairly simple and/or low-volume transactions
- ▲ Q/LinQ can replace CIM and improve on many built-in limitations in many respects

Topic Summary

- ▲ Use CIM to mass load data into database
- ▲ Testing of CIM data is vital to ensure data integrity

Exercise: CIM Lab



CIM Lab Exercise (20 minutes)

- ▲ Manually build a CIM file to load a new customer into Customer Maintenance (adcsmt.p) or a new item into Item Master Maintenance (ppptmt.p)

CIM Lab Solution

- ▲ This is the same program that we walked through in the lecture session
- ▲ Depending on what settings QAD Enterprise Applications has, the CIM file in the CIM interface chapter should be the solution



CIM Lab Solution (Customer Maintenance)

@@batchload adcsmt.p

"mot-004"

"Motorola" "1303 East Algonquin Road" - - "Schaumburg" "IL" ~

"60196" "0" "USA" - "Bob Galvin" "555-867-5309" "0000"

"Second Sort Name" - no "UPS" - - - - "was customer # MOT" - - ~

- "10000" -

"MV"

no

1000000

"UPS-RED" 5 "prepaid"

.

.

@@end



CIM Lab Solution (Item Master Maintenance)

@@batchload pptmt.p

"item-1"

"EA" "CIM loading 1.4.1" "description2"

"1000" - - - "RAW" "AC" "FINGOOD"

- - "10000"

-

-

10

-

-

.

@@end

QAD Proprietary

BPC-14-CIM-470

- Save the above text into '01-04-01.txt'
- Use CIM Data Load to load data from the text file '01-04-01.txt'
- This will create records in bdl_mstr (Batch Data Load Master) and bdld_det (Batch Data Load Detail)
- Use CIM Data Load Processor to load the information from bdl_mstr and bdld_det to the appropriate table (that is, creates an item in pt_mstr)

Review



CIM Interface

- ✓ CIM Mechanism
- ✓ Rationale for using CIM
- ✓ CIM Data File
- ✓ CIM Test Program
- ✓ Loading CIM Data Files
- ✓ Processing CIM Data Files

Chapter 15

E-Mail



E-Mail

In this section, you learn about:

- ▲ Setup and use of e-mail in QAD Enterprise Applications

E-Mail Definition Maintenance

E-mail Definition Maintenance [mgemmt.p]

E-mail System: DemoMail
 Operating System: Unix
 Start Effective: 11/20/2008 End Effective:
 Description:
 Path and Program Name:
 Command Line Parameters:
 Sender: Sequence:
 Recipient: Sequence:
 Subject: Sequence:
 Message Text File: Sequence:
 Message Text String: Sequence:
 Start Effective: End Effective:
 E-Mail Command:
 Back Next

QAD Proprietary E-Mail Definition Maintenance (36.4.20) BPC-15-EM-02

To send e-mail from QAD Enterprise Applications:

- 1 Copy wmailto.exe into C:\qad\flysim\work86
Note See #4 below if wmailto is copied into anything other than C:\qad\flysim\work86
- 2 Initialize wmailto / exit to DOS / run wmailto -init
 - Enter your mail server (e.g. oh-mail) / enter your default return path (e.g. pma)
 - All other requested inputs should be left blank
- 3 Two e-mail definitions have been provided in E-mail Definition Maintenance:
 - DemoMail for Unix
 - DemoMail for Win32
- 4 If wmailto has not been put in the default directory:
 - Select DemoMail for Win 32
 - Change the Path and Program Name field point to the directory where wmailto has been placed
- 5 To verify your setup login as yourself:
 - Go to E-mail Definition Maintenance
 - Bring up DemoMail for win32

- F2 through the record
- When it asks if you want to send a test message, select yes

User Maintenance

The screenshot shows the 'User Maintenance' window for user 'mdf' (Mark Farley). The window includes a header with the QAD logo and the title 'User Maintenance [mgcn023.w]'. Below the header, there are tabs for 'Go To' and 'Action'. The main form contains the following fields and options:

- User ID: mdf
- User Name: Mark Farley
- Language: us (with a globe icon)
- Country Code: usa (with a globe icon)
- User Type: Employee (with a magnifying glass icon)
- Time Zone: PST/PDT (with a magnifying glass icon)
- E-mail Def: DemoMail (with a magnifying glass icon)
- E-mail Address: mdf@qad.com
- Menu Style: A (with a dropdown arrow) (A - Icons B - Tear Off C - Character)
- Menu Substitution: (with a checkbox)
- Remark: (with a text area)
- Variant: (with a text field)
- Restricted: (with a checkbox)
- Access Location: (with a magnifying glass icon)

Below the main form is the 'System Access' section, which includes:

- Active: ☒
- Active Reason: (with a text field)
- Force Password Change: ☒
- Update Password: ☒
- Last Logon: (with a dropdown arrow) 00:00
- Last Password Change: (with a dropdown arrow)

At the bottom right of the form are three buttons: 'Delete', 'Back', and 'Next'. The footer of the window displays 'QAD Proprietary', 'User Maintenance (36.3.1)', and 'BPC-15-EM-03'.

1 Go to User Maintenance

- Set up your Userid and Language (for example, Userid is pma / Language is US)

2 Go to the e-mail folder and enter your e-mail address (for example, pma@qad.com)

- Enter Description: DemoMail

To run e-mail, you will need to be connected to the network and logged in as yourself.

Example



Program Example

```
/* xxmail.p - STUB PROGRAM TO SEND E-MAIL */
/*V8:ConvertMode=Maintenance*/
{mfdeclre.i}

define variable subject as character no-undo.
define variable deliver_to as character no-undo.
define variable message_text as character no-undo.

/* BUILD SUBJECT LINE */
subject = "".

/* BUILD COMMA DELIMITED DISTRIBUTION LIST */
deliver_to = "".

/* BUILD MESSAGE */

message_text = "".
run proc_send_mail(input subject,
    input deliver_to,
    input message_text).
```

Internal Procedure



Internal Procedure

```

output_filename = mfguser + ".xxe".
output to value(output_filename).
display p_message with frame cmmt.
output close.

do v_count = 1 to num-entries(p_delete)
  {gprun.i ""mgemsend.p""
    "(input "",
      input entry(v_count,
      input p_subject,
      input output_filename,
      input "",
      input false,
      input ?,
      output_email_return_code,
      output_email_return_code)
    if email_return_code <> 0 then
      {mfmsg.i email_return_code 2,
    end.
  end.
os-delete value(output_filename).
  
```

recipient (usr_userid)
 e-mail address
 subject of message
 message filename
 message string
 TEST MODE flag
 recid of e-mail definition
 record return code
 user id in error, if any

Event Notification



Event Notification

- ▲ Built into the PCC Module
 - Notify interested parties when a PCO is submitted for approval
 - Notify the next person on the approval routing
 - Notify interested parties when a PCO has been approved
- ▲ Incorporated into the GRS Design
 - Notify the requestor when status of a requisition has changed
 - Notify the next person on the approval routing
 - Notify Purchasing when the requisition has been approved

QAD Proprietary

BPC-15-EM-064

Be sure to note that GRS is “Future” either 8.6D or *.7 and is not in current release.

Other Possibilities



Other Possibilities

- ▲ Use with the component configurator
 - Send e-mail to Costing Department when a new item is added
- ▲ Use as toolbar button
 - Send e-mail to next person in workflow
- ▲ Use with database triggers
 - Send e-mail to Purchasing when inventory falls below reorder point

QAD Proprietary

BPC-15-EM-07

```
in_mstr Delete no ind.t Write no inw.t
in_qty_oh ASSIGN no D:\MFG86C\Gui\triggers\Twintrig.p
```

This trigger monitors quantity on-hand against reorder point and displays a message whenever an inventory transaction drives QOH below reorder.

Database triggers are executed whenever the specified event occurs, whether in normal mode, in batch load, or as a background process. It is therefore not a good idea to include user interaction in a database trigger. The trigger must, therefore, be intelligent enough to build its own Subject, Deliver-to, and message-text. Essentially, this is what is done in the PCC module where the deliver-to's are kept in a table and the subject and message are hard coded based on if...then...else.

Example



Trigger Example

```

/* Monitor Re-Order value, if greater than      */
/* On-Hand + Allocated, Signal User to reorder */

define variable tw_rop      like pt_rop      no-undo.
define variable qty_to_ord like in_qty_oh no-undo.

for first pt_mstr where pt_part = in_part no-lock:
end.
tw_rop = pt_rop.

if pt_site <> in_site and
  can-find( ptp_det where ptp_part = in_part and
    ptp_site = in_site) then do:
  for first ptp_det
    where ptp_site = in_site:
  end.
  tw_rop = ptp_rop.
end.

if tw_rop > in_qty_oh then do:
  qty_to_ord = tw_rop - in_qty_oh.
  message "WARNING - Quantity Below Reorder Point!"
    "Please place an order for" pt_mstr.pt_desc1
    ". Item Number: " pt_mstr.pt_part
    "Order quantity at least: " qty_to_ord
    view-as alert-box information buttons ok.
end.

```

Trigger Example

```

/* Monitor Re-Order value, if greater than */
/* On-Hand + Allocated, Signal User to reorder */

define variable tw_rop like pt_rop no-undo.
define variable qty_to_ord like in_qty_oh no-undo.

for first pt_mstr where pt_mstr.site = in_site do
end.
tw_rop = pt_rop.

if pt_site <> in_site then
  can-find(ptp_det
  ptp_site = in_site
  for first ptp_det
  where ptp_site = in_site
  end.
  tw_rop = ptp_rop
end.

if tw_rop > in_qty_to_ord then
  message "WARNING
  "Please place
  ". Item Number: " + pt_mstr.pt_part
  "Order quantity at least: " +
  qty_to_ord"
  run proc_send_mail(input subject,
  input deliver_to,
  input message_text).
  view-as alert-box information buttons ok.
end.

```

Subject = "WARNING - Quantity below Reorder Point!"
Deliver_to = "purchasing@company.com"
Message_text = "Please place an order for "
 + pt_mstr.pt_desc1
 + ". Item Number: " + pt_mstr.pt_part
 + " Order quantity at least: " +
 qty_to_ord"

run proc_send_mail(input subject,
 input deliver_to,
 input message_text).

Chapter 16

International Multi-Language Labels



International Multi-Language Labels

In this section, you learn about:

- ▲ Characteristics of label making
- ▲ Label Master Terms & Label Detail
- ▲ getTermLabel function and its variations
- ▲ getFrameLabel function
- ▲ Using Message Master
- ▲ Using Master Comments

General Translation Principles



General Translation Principles (1 of 3)

- ▲ All displayable text in a program must come from **msg_mstr, lngd_det, lbl_mstr or mnt_det**
 - messages come from msg_mstr
 - strings that require mnemonics come from lngd_det
 - labels come from lbl_mstr
 - menu titles come from mnt_det.
- ▲ All labeled frames must have a **setFrameLabels()** call
- ▲ All literal strings **must** be removed from frame definitions
- ▲ All fields that display a label within a frame that uses setFrameLabels() must be supported by Label Detail records



General Translation Principles (2 of 3)

- ▲ Label master and detail records are release specific
(starting with eB)
- ▲ New Label master record needed for new tables and fields added to the schema
- ▲ Label detail records required linking new schema field with label stored in label term table
- ▲ All explicit use of label-retrieving functions must be supported by Label Master records

QAD Proprietary

BPC-16-MLL-030

- For all new tables and fields added to the schema, new label master (if needed) and label detail records are required to link each new schema field with the desired label stored in the label term table
- All explicit use of label-retrieving functions [such as `getTermLabel()` and `getFrameTitle()`] must be supported by Label Master records

General Translation Principles (3 of 3)

- ▲ Do not build sentences from separate text strings
- ▲ Create new messages in msg_mstr for displaying informational messages, errors and warning messages
- ▲ Leave plenty of real estate for a text string to expand in
 - Allow 30% for strings > 10 characters
 - Allow 100% for strings ? 10 characters
- ▲ Use standard abbreviations, spellings, grammar and terminology

Accessing Label Functions



Accessing label functions within a program: Use of gplabel.i and gplabel.p (1 of 2)

gplabel.i

- ▲ Contains definitions of all label-retrieving functions
- ▲ Must be included in each .p file that uses a label function
- ▲ It is included in mfdtitle.i
- ▲ If program does not include mfdtitle.i, gplabel.i should be included near top of the file (after the revision tags and mfdeclre.i call)
- ▲ If mfdeclre.i is not included in the program,
define shared variable global_user_lang_dir...
is required

QAD Proprietary

BPC-16-MLL-054

- Any program that includes mfdtitle.i does not need to include gplabel.i since it is included in mfdtitle.i



Accessing label functions within a program: Use of gplabel.i and gplabel.p (2 of 2)

gplabel.p

- ▲ Responsible for displaying the labels
 - Searches best fit and starts from long-label. If no fit then substring long label.
- ▲ Persistent procedure

Label Master Maintenance

**QAD**
Our Passion. Your Advantage.

QAD Enterprise Applications Function: Label Master Maintenance

Label Master Maintenance x

Go To Action

Language ID: us english (U.S.)
Term: SALES_ORDER

Long Label: Sales Order

Medium Label: Order

Short Label: SO

Stacked Label: Sales!Order

Description:

Delete Back Next

QAD Proprietary

Label Master Maintenance (36.4.17.1)

BPC-16-MLL-07

Label Master (lbl_mstr)

- ▲ The Label Master table (lbl_mstr) stores all labels that appear on QAD Enterprise Applications's screens and reports

Term	35 chars	<i>SALES_ORDER</i>
Language		<i>US</i>
Long label	50 chars	<i>Sales Order</i>
Medium label	30 chars	<i>Order</i>
Short label	20 chars	<i>SO</i>
Stacked label	50 chars	<i>Sales!Order</i>

Term: meaning of a label

Language: language the record's labels appear in

- gplabel.p determines best fit.

Term Naming Conventions



Term Naming Conventions

- ▲ A term must be in upper case
- ▲ Cannot contain any spaces. Spaces need to be replaced with underscores.
- ▲ Cannot contain non-alphabetic/non-numerical characters like !,&, / etc
- ▲ European Accounting labels must start with EA_
- ▲ MLP labels must start with ML_
- ▲ TIPAC labels must start with TP_

Label Master Terms



Label Master Terms

- ▲ Acceptable terms:
 - SALES_ORDER_NUMBER
 - CUSTOMER_ADDRESS
 - VALUE_ADDED_TAX



Label Master Terms

- ▲ LINE
- ▲ CHECK
- ▲ POST
- ▲ ORDER
- ▲ PAGE
 - Ambiguous terms are not acceptable

Rules



Rules for the long, medium, short, and stacked labels

- ▲ Must be in English
- ▲ Must be in mixed case
- ▲ Must have each principal words begin with a capital letter
- ▲ Begin and end with alphanumeric characters
- ▲ Must not contain two or more consecutive white spaces
- ▲ Must contain only alphanumeric characters and white spaces (no punctuation marks)
- ▲ Be at least 2 characters long and no more than 45 characters long



Rules for the long, medium, short, and stacked labels

- ▲ Must contain only standard abbreviations and spellings
- ▲ Must not be used to combine with other labels to form longer strings
- ▲ Must be unambiguously associated with a Label Master term

Example:

Term:	"SALES_ORDER"
Long Label:	"Sales Order"
Medium Label:	"Order"
Short Label:	"SO"
Stacked Label:	"Sales!Order"

Incorrect

"CONTINUED"
"Item description"
" Customer Bank "
"Qty Date
 Description"
"Qty. Billed"
"Total:"
"User Id"
"Directory not found."

Correct

"Continued"
"Item Description"
"Customer Bank"
"Qty", "Date", and
 "Description"
"Qty Billed"
"Total"
"User ID"
(Belongs in msg_mstr.)

 QAD Our Passion. Your Advantage.	
Incorrect	Correct
"Current session for "	Untranslatable.
"EU" (for <i>End User</i>)	No abbreviation should be used
"Lbcd" (for <i>Labor Code</i>)	No abbreviation should be used
"Fr" (for <i>Freight</i>)	No abbreviation should be used
"T" (for <i>Type</i>)	If you must abbreviate "Type," the layout of the screen or report is probably bad
"S/O Maintenance"	"SO Maintenance"

QAD Proprietary

BPC-16-MLL-150

Label Detail Maintenance

 QAD
Our Passion. Your Advantage.

QAD Enterprise Applications Function: Label Detail Maintenance

Label Detail Maintenance X

Go To Action

Field Name: sob_nbr
Program Name:
Term: SALES_ORDER
Long Label: Sales Order
Medium Label: Order
Short Label: SO
Stacked Label: Sales!Order

DeleteBackNext

Label Detail Maintenance (36.4.17.5)

QAD Proprietary BPC-16-MLL-166



Label detail (lbld_det)

- ▲ Label Detail table (lbld_det) associates each term (defined in lbl_mstr) with all of the fields and variables in the schema and source code that use that term's labels

Label detail (lbld_det)

- ▲ Label Detail table will allow a term to be associated with a variable or field for a particular menu-level program

General

- Fieldname *so_nbr*
- (top-level) program
- Term *SALES_ORDER*

Exceptions

- Fieldname *so_nbr*
- (top-level) program *sosorp05.p*
- Term *REFERENCE*



How to Retrieve a Label Using Functions (gplabel.p)



QAD Proprietary

BPC-16-MLL-190

Function: `getTermLabel()`

- ▲ Returns a label for a specified term
- ▲ Takes 2 parameters: term, maximum length
- ▲ Returns a character string (to be used as a label)
- ▲ Determines which label to return (long, medium, short) based on the maximum length specified
- ▲ Returns the term itself if no label is found



Function: **getTermLabel()**

Example1: `getTermLabel ("CUSTOMER_ADDRESS", 30)`

Returns: "Customer Address"

Example2: Given the following lbl_mstr record

Term: "**SALES_ORDER**"

Long Label: "**Sales Order**"

Medium Label: "**Order**"

Short Label: "**SO**"

getTermLabel("SALES_ORDER",10)

returns: "**Order**"

getTermLabel("SALES_ORDER",4)

returns: "**SO**"

QAD Proprietary

BPC-16-MLL-210

Example1

The length parameter sets the limit on the string's length, but does not determine how long the returned string is. The length of the string is that of the longest available label in lbl_mstr (in this example, 16 bytes) for the specified term that does not exceed the length parameter (in this example, 30 bytes).



Variations of `getTermLabel()` `getFrameTitle()`

Example:

```
getFrameTitle("CUSTOMER_ADDRESS", 30)
```

Returns:

```
" Customer Address "
```

(Note the single white space in the leading and trailing positions)

QAD Proprietary

BPC-16-MLL-220

- **`getFrameTitle()`** returns a frame title for a specified term. A frame title is a string with a single white space in both the string's lead and trail positions. Same input parameters as `getTermLabel()`.
- As with `getTermLabel()`, the string's overall length is restricted by the length parameter (in this example, 30) but is not wholly determined by it.



Variations of `getTermLabel()` `getFrameTitle()`

Example, part 2:

```
form
  csbd_bank
  bk_acct_type
with down frame csbd width 80 title color
normal
  (getFrameTitle("BANK_ACCOUNTS_DATA",78)).

/* SET EXTERNAL RECORDS */
setFrameLabels(frame csbd:handle).
```




Variations of `getTermLabel()`

`getTermLabelRt()`

Example:

```
getTermLabelRt("CUSTOMER_ADDRESS",20)
```

Returns:

```
"      Customer Address"
```

QAD Proprietary

BPC-16-MLL-240

- `getTermLabelRt()` returns a right justified label for a specified label term. Same input parameters as `getTermLabel()`.



Variations of `getTermLabel()`

`getTermLabelRtColon()`

Example:

```
getTermLabelRtColon("CUSTOMER_ADDRESS", 20)
```

Returns:

```
"    Customer Address:"
```

QAD Proprietary

BPC-16-MLL-250

- `getTermLabelRtColon()` returns a right justified label with an appended colon for a specified term. Same input parameters as `getTermLabel()`.



Variations of `getTermLabel()`

`getTermLabelCentered()`

Example:

```
getTermLabelCentered("CUSTOMER_ADDRES  
S", 30)
```

Returns:

```
"          Customer Address          "
```

QAD Proprietary

BPC-16-MLL-260

- `getTermLabelCentered()` returns a label for a specified term, centered within the specified length. Same input parameters as `getTermLabel()`.



Variations of `getTermLabel()`

`getTermLabelFillCentered()`

Example:

```
dynamic-function('getTermLabelFillCentered'
  in h-label, Input "CUSTOMER_ADDRESS",
      Input 30,
      Input "*")
```

returns:

```
"***** Customer Address *****"
```

QAD Proprietary

BPC-16-MLL-270

- `getTermLabelFillCentered()` returns a centered label, the head and trail of which is filled with any specified character
 - Same input parameters as `getTermLabel()`, plus an additional parameter, `p-fillchar` (the character with which to fill the string). *As this function is not defined in `gplabel.i`, it must be called as a dynamic function.*
 - This function leaves a single space between the text and the fill-in character
 - The function will always return a string that contains text and at least one lead and trail instance of the fill-in character

Setting a Frame Label

- ▲ FUNCTION: setFrameLabels()
- ▲ Assign all labels in frames with foreign language labels (side or column)
- ▲ Takes frame handle as input
- ▲ Uses (behind the scenes) all the label-retrieving functions in gplabel.p

QAD Proprietary

BPC-16-MLL-280

- In most cases, it is unnecessary to explicitly invoke the label-retrieving functions listed before
- For most frames it is sufficient to make a single call to setFrameLabels(), passing it the frame's handle.
 - setFrameLabels() goes through the widget tree of a frame, determining what fields belong to the frame and calculating the real estate available for each field's label
 - setFrameLabels() then sets the correct language version of the label required for each field
- setFrameLabels figures out the layout of a frame by the use of two of its internal procedures, setSideLabels and setColumnLabels. Each of these procedures operates with a completely different set of algorithms.
 - setColumnLabels is concerned with counting the number of columns and figuring the real estate (based on a field's format) for each of the columns
 - setSideLabels figures the real estate by taking the width of the individual widgets per row

Function: setFrameLabels()

form

```
    wr_nbr          colon 24
    wr_lot          colon 50
    wr_part         colon 24
    pt_desc1        no-label
    wr_op           colon 24
    wr_std_op       colon 24
with frame a side-labels
/* SET EXTERNAL LABELS */
width 80 no-attr-space.
setFrameLabels(frame a:handle).
```

How To: Placement of setFrameLabels()

- ▲ All displayed frames should have setFrameLabels() except for no-label frames, header frames, and frames in which all fields are defined as no-label
- ▲ “buffer” frames which don’t typically display, should also NOT have a setFrameLabels() call
- ▲ The placement of setFrameLabels() should be guided by the idea that all labeled frames should have a setFrameLabels() call within scope of the frame

QAD Proprietary

BPC-16-MLL-300

- A header frame is one that includes the keyword HEADER after the FORM statement.
- A no-label frame is one that includes the NO-LABEL qualifier in the frame definition line (typically the line that contains the “WITH frame <framename>” phrase).
- A buffer frame is a frame that contains a table or buffer variable only.

How To: Placement of setFrameLabels()

▲ In general:

- each displayed frame must be named
- each named frame needs to have only one call to setFrameLabels()
- each setFrameLabels() call must appear within the scope of the named frame

How To: Placement of setFrameLabels()

Example 1:

- ▲ If the frame is defined by a FORM statement, insert setFrameLabels() directly after the last line of the "with frame" statement:

```
form
    so_nbr colon 15
with frame a side-labels width 80.

/* SET EXTERNAL LABELS */
setFrameLabels(frame a:handle).
```



How To: Placement of setFrameLabels()

Example 2:

- ▲ If the frame is defined within a FOR EACH loop, insert setFrameLabels() within the loop.

```
for each x with frame b:
    statement.
    /* SET EXTERNAL LABELS */
    setFrameLabels(frame b:handle) .
    display x.
end.
```

QAD Proprietary

BPC-16-MLL-330

Note This example only applies for frames that are defined within a FOR EACH loop. Most frames, of course, are defined before the FOR EACH loop and so don't require setFrameLabels() within the loop.

How To: Placement of setFrameLabels()

Example 3:

- ▲ If the frame is defined within the branch of a conditional, insert setFrameLabels() within the conditional branch

```
if ... then do with frame b:  
    /* SET EXTERNAL LABELS */  
    setFrameLabels(frame b:handle).  
    display x.  
end.  
  
and  
  
else ... do with frame b:  
    /* SET EXTERNAL LABELS */  
    setFrameLabels(frame b:handle).  
    display x.  
end.
```

Where the frame is defined in the conditional display, the above constructs should be used.



How To: Placement of setFrameLabels()

- ▲ For frames not defined prior to the display, the following constructs should be avoided:

and if ... then display x.
 else ... display x.

How To: Placement of setFrameLabels()

Example 4:

- ▲ If the frame is defined within a display statement but outside of a loop, create a DO loop and insert setFrameLabels() within the loop

display x with frame b.

should be rewritten as:

```
do with frame b:  
    /* SET EXTERNAL LABELS */  
    setFrameLabels(frame b:handle).  
    display x.  
end.
```



How To: Frames That Contain Button Widgets

- ▲ Buttons should be defined as “side label” type widgets
- ▲ Progress allows Buttons in a down frame
- ▲ They should only be defined within side-label frames

```
define frame f-button
    b-end at 1
    with no-box overlay three-d side-labels width
    10.
```

```
/* SET EXTERNAL LABELS */
setFrameLabels(frame f-button:handle).
```

QAD Proprietary

BPC-16-MLL-370

- Buttons should be defined as “side label” type widgets.
- Progress allows them to be used in a down frame, but, because of the algorithms used in setFrameLabels, they should only be defined within side-label frames.



How To: Handling the Format of Logical Variables

- ▲ The format for all logical variables must be defined as a term in the Label master table

For example:

```
define variable StaCmf           like mfc_logical
                                format "Accepted/Rejected" no-undo.
```

The following Label master record must be defined:

Term: **"ACCEPTED/REJECTED"**
Long Label: **"Accepted/Rejected"**
Medium Label: **"**
Short Label: **"A/R"**
Description: **Logical Format**



How To: Handling Character Variables That Contain Literal Text

- ▲ Do not specify an initial value in the DEFINE statement for character variables that subsequently will be displayed in a frame. Instead, set the initial value for character strings using the getTermLabel function:

Example:

```
define variable del as character format "x(15)".  
del = getTermLabel("DELETED",15).  
/* ASSIGN INITIAL VALUE */
```

- ▲ If a FORM statement contains a character array that displays any or all of its extents as separate character strings, use getTermLabel() for each extent of the variable



How To: Handling Character Variables That Contain Literal Text

Example:

```
define new shared variable sortextoption as character extent 2 format "x(58)".
assign sortextoption[1] = "1 - " + getTermLabel("BY_SHIP-FROM",12) + ", " +
                        getTermLabel("CUSTOMER",8) + ", " +
                        getTermLabel("SHIP-TO",7)
sortextoption[2] = "2 - " + getTermLabel("BY_SHIP-FROM",12) + ", " +
                        getTermLabel("ITEM",4) + ", " +
                        getTermLabel("CUSTOMER",8).
```

form

```
shipfrom_from    colon 15 label "Ship-From"
shipfrom_to      colon 45 label {t001.i}
cust_from        colon 15
cust_to          colon 45 label {t001.I}
part_from        colon 15
part_to          colon 45 label {t001.i} skip
sortextoption[1] at 17 no-label
sortextoption[2] at 17 no-label skip(1)
with frame a side-labels width 80 attr-space.
```

```
setFrameLabels(frame a:handle). /* SET EXTERNAL LABELS */
```



How To: Handling Character Variables That Contain Literal Text

- ▲ If the FORM or frame contains a string that has been set by the ENTRY function, use getTermLabel() for each such string.

Example:

define variable disp-wdays as character no-undo.

```
disp-wdays = getTermLabel("SUNDAY",12) + "," +  
              getTermLabel("MONDAY",12) + "," +  
              getTermLabel("TUESDAY",12) + "," +  
              getTermLabel("WEDNESDAY",12) + "," +  
              getTermLabel("THURSDAY",12) + "," +  
              getTermLabel("FRIDAY",12) + "," +  
              getTermLabel("SATURDAY",12).  
display entry(weekday(last_date),disp-wdays).
```



How To: Handling Character Variables That Contain Literal Text

- ▲ You cannot use a user defined function (such as `getTermLabel`) within an `ASSIGN` statement after the assignment of the key fields

Wrong:

```
create rnd_mstr.
```

```
assign
```

```
    rnd_rnd_mthd = "0"
```

```
    rnd_unit = 1
```

```
    rnd_thrshld = 0.5
```

```
    rnd_dec_pt = "."
```

```
    rnd_desc =
```

```
getTermLabel("ROUND_TO_ONES",16).
```

**The above will generate an error since
rnd_rnd_mthd is a key field!**



How To: Handling Character Variables That Contain Literal Text

To avoid this problem, either:

1. Move all key field assignments after all User-defined Function references in the ASSIGN statement as follows:

```
assign rnd_desc =  
    getTermLabel("ROUND_TO_ONES",16)  
        rnd_unit = 1  
        rnd_thrshld = 0.5  
        rnd_dec_pt = "."  
        rnd_rnd_mthd = "0".
```

OR...



How To: Handling Character Variables That Contain Literal Text

2. Do all key field assignments in a separate ASSIGN statement:

```
assign rnd_unit = 1

rnd_thrshld = 0.5
rnd_dec_pt = "."

rnd_desc =
getTermLabel("ROUND_TO_ONES",16) .
assign rnd_rnd_mthd = "0".
```



How To: Handling Literal Strings in Frames

- ▲ To display a literal string (i.e., a text string not associated with a field or variable) in a no-label, side-label or down frame, define it as a character variable, set the initial value using `getTermLabel()`, reference the variable name in the FORM or DISPLAY statements.

The NO-LABEL qualifier should be specified in either the DEFINE statement or subsequent FORM statement.

How To: Handling Literal Strings in Frames

Example:

```
define variable c-shipment as character format
"x(19)" no-undo.
c-shipment = getTermLabel("SHIPMENT",8) + fill("-",
", 11).
form
    c-shipment    no-label    at 87
                  sbi_xmission    colon 16
                  sbid_qty        colon 60
with frame f_detail side-labels width 132 no-box
no-attr-space down.

/* SET EXTERNAL LABELS */
setFrameLabels(frame f_detail:handle).
display
    c-shipment
        sbi_xmission
        sbid_qty
with frame f_detail.
```



How To: Passing Literal Strings as Arguments to Include Files

Include File	Literal String Argument	How literal is used in Include file	Example
mfadform.i	{3} frame title	(getFrameTitle("{3}",25))	sosomtp.p {mfadform.i "sold_to" 1 SOLD-TO }
mfmrw.i, mfmrwnew.i, mfmrw01.i	{10} detail	if "{10}" > " " then mrp_det.mrp_detail = getTermLabel("{10}",24).	dsdmmmt01.p {mfmrw.I "ds_det", ds_part ds_req_nbr ds_shipsite ds_site ? ds_shipdate qty_open "DEMAND" INTERSITE_DEMAND ds_shipsite}

Note: Any new include files which require literal strings for arguments should be implemented in a similar fashion.

How To: Display Frame Titles

- ▲ Use getTitle() to display the frame title
- ▲ Specify the maximum length possible for the frame title

Dynamic Frame Titles:

Dynamic frame titles are those that change during program execution depending on the information being displayed. Create one or more variables to store the frame title and use getTitle to retrieve the desired frame title.



How To: Display Frame Titles

Example:

```
define variable frametitle-so as character
format "x(20)".
define variable frametitle-rma-iss as
character format "x(20)".
assign
    frametitle-so =
    getFrameTitle("SALES_ORDER_LINE",20)
    frametitle-rma-iss =
    getFrameTitle("RMA_ISSUE_LINE",20).
if this-is-rma then
    if rma-issue-line then
        frametitle = frametitle-rma-iss.
else
    frametitle = frametitle-so.
```



How To: Fields That Have Multiple Labels Within a Single Menu Item

Example 1:

```

if base_rpt = "" and et_report_curr = base_curr
then
  assign
    l_label1 = getTermLabelRtColon("BASE_SUPPLIER_TOTALS",25)
    l_label2 = getTermLabelRtColon("BASE_REPORT_TOTALS",25) .
else
  assign
    l_label1 = string(et_report_curr,"x(3)") + " "
               + getTermLabelRtColon("SUPPLIER_TOTALS",21)
    l_label2 = string(et_report_curr,"x(3)") + " "
               + getTermLabelRtColon("REPORT_TOTALS",21) .

```

QAD Proprietary

BPC-16-MLL-500

Use different variable names when you need to display the same field in different frames with different labels.



How To: Fields That Have Multiple Labels Within a Single Menu Item

▲ Example 2:

if eximmode = {&IMPORT} then run qqimp in this-procedure.
 else if eximmode = {&EXPORT} then run qqexp in this-procedure.

PROCEDURE qqimp:

form

qqGood format "zzz,zz9" colon 70 label {&qqierp_p_17}

skip

qqWarn format "zzz,zz9" colon 70 label {&qqierp_p_18}

skip

with frame ImpStats side-labels width 150.

/* SET EXTERNAL LABELS */

setFrameLabels(frame ImpStats:handle).

END PROCEDURE. /* qqimp */

QAD Proprietary

BPC-16-MLL-510

If there are fields in the frame that appear in another frame in the same program but have different labels, use setFrameLabels() for the first frame and use getTranslateFramesFlag() for subsequent frames.

Example 2 continued:

PROCEDURE qqexp:

form

qqGood format "zzz,zz9" colon 70 label {&qqierp_p_22}

skip

**qqWarn format "zzz,zz9" colon 70 label {&qqierp_p_23} skip
with frame ExpStats side-labels width 150.**

**/* DIFFERENT LABELS ARE DISPLAYED FOR THE SAME
VARIABLE */**

/* SO NEED TO ASSIGN LABELS EXPLICITLY. */

if dynamic-function('getTranslateFramesFlag' in h-label) then

assign

**qqGood:label = getTermLabel(
"NUMBER_OF_DOCS_SENT_SUCCESSFULLY", 68)**

**qqWarn:label = getTermLabel(
"NUMBER_OF_DOCS_SENT_WITH_WARNINGS", 68)**

END PROCEDURE.



How To: Handle PUT Statements

Example 1:

```
put
    "-----" to 107
    "-----" to 132
    skip(0)
    string(et_report_curr + " " +
    getTermLabel("SALESPERSON_TOTAL",60)
+ ":" )
    format "x(25)" to 70.
```

QAD Proprietary

BPC-16-MLL-530

A PUT statement requires that a format be passed to it for the string.

Note getTermLabel within gplblfmt.i requires two pairs of double quotes.



How To: Handle PUT Statements

Example 2:

```
if multi-due then  
  put invoice at 1  
    {gplblfmt.i &FUNC=getTermLabel("MULTIPLE",10) } at 14.
```

QAD Proprietary

BPC-16-MLL-540

- If length available for the string is not fixed, use a dynamic calculation of the string's length
- This allows a string to be longer or shorter, depending upon its translation
- Use gplblfmt.i, which calculates the format of the string based on the label retrieved and uses that for the format length for the PUT statement



How To: Displays With ACCUM, SUM, AVG or Similar Keywords

For any calculations such as ACCUM, SUM and AVG where you wish to display the resulting calculation with a label:

- ▲ Define variables to store the calculations being performed.
- ▲ Display the variables. This allows setFrameLabels() to handle the frame.
- ▲ The label for the variable can be defined in either the DEFINE or DISPLAY statement.



How To: Displays With ACCUM, SUM, AVG or Similar Keywords

Example:

```
define variable std_hrs_st  as decimal no-undo.
define variable prod_hrs_st as decimal no-undo.
define variable prod_pct_st as decimal no-undo.
```

assign

```
std_hrs_st = (accum sub-total by op_emp std_hours)
prod_hrs_st = (accum sub-total by op_emp prod_hours)
prod_pct_st = 100 * std_hrs_st / prod_hrs_st.
```

display

```
prod_pct_st label "Prod %"  format "->>9.9" space(2)
prod_hrs_st label "Prod Hrs" format "->, >>9.9"
std_hrs_st  label "Std Hrs"  format "->, >>9.9"
with frame b.
```



Using Message Master

- ▲ Labels > 50 character and < 60 should go via the message master (thus using the `mfmsgnn.i` or `pxmsg.i` include)

```
/*ebprac03.p: Use of label function for 50 > label > 60*/
```

```
{mfdeclre.i}
```

```
define variable l_msg as character.
```

```
{pxmsg.i &MSGNUM=123
```

```
    &MSGBUFFER=l_msg}
```

```
message "l_msg:" l_msg.
```

```
l_msg: End date cannot be before start date
```

Using Master Comments

- ▲ Labels > 60 characters, use master commands
 - Multiple lines allowed
- ▲ When you need to include a paragraph (or several sentences of text) in a program, such as a utility program, you should use the include file {gpcdget.i} and enter the text in Master Comments
- ▲ **gpcdget.i** should be included prior to the form/display statement for any input frame in the source file of a utility program



Example: : utbrgnmt.p

```

/* DISPLAY TEXT FRAME*/
{gpcdget.i "UT"}

/*****
*      form
*      {t081.i}
*      with frame aa width 80.
*
*      /* SET EXTERNAL LABELS */
*      setFrameLabels(frame aa:handle).
*****/

/* INPUT SELECTION FRAME */
form
  skip(1)
  browse1    colon 25
  browse2    colon 45 label {t001.i}
  with frame a side-labels width 80.

/* SET EXTERNAL LABELS */
setFrameLabels(frame a:handle).

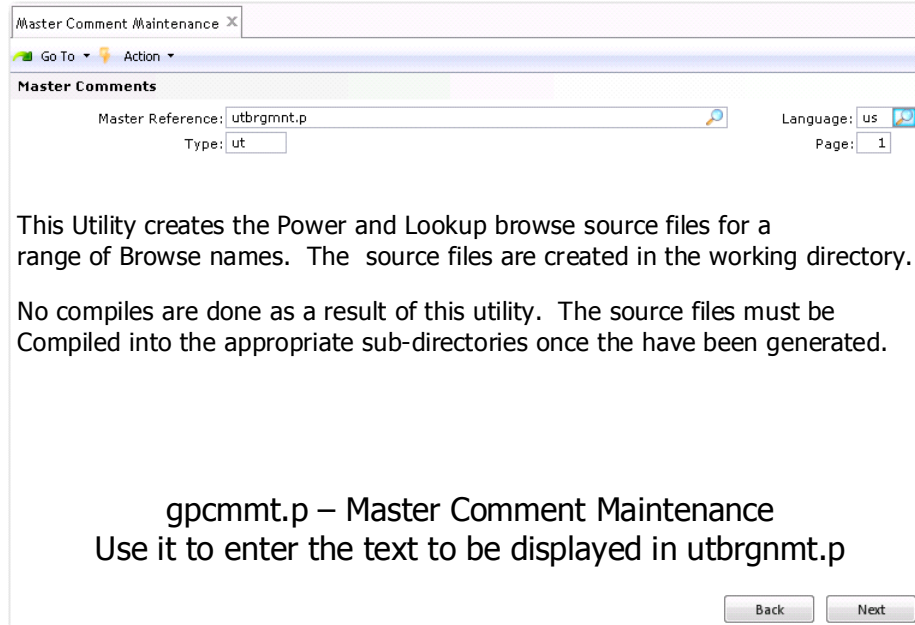
```

QAD Proprietary

BPC-16-MLL-590

The “UT” parameters passed to gpcdget.i is used as part of the key to the table cd_det. It is used for all utility programs (ut*.p and ux*.p). When the “UT” is used with the include file, the text is retrieved and displayed.

Example: Master Comments



Master Comment Maintenance

Go To Action

Master Comments

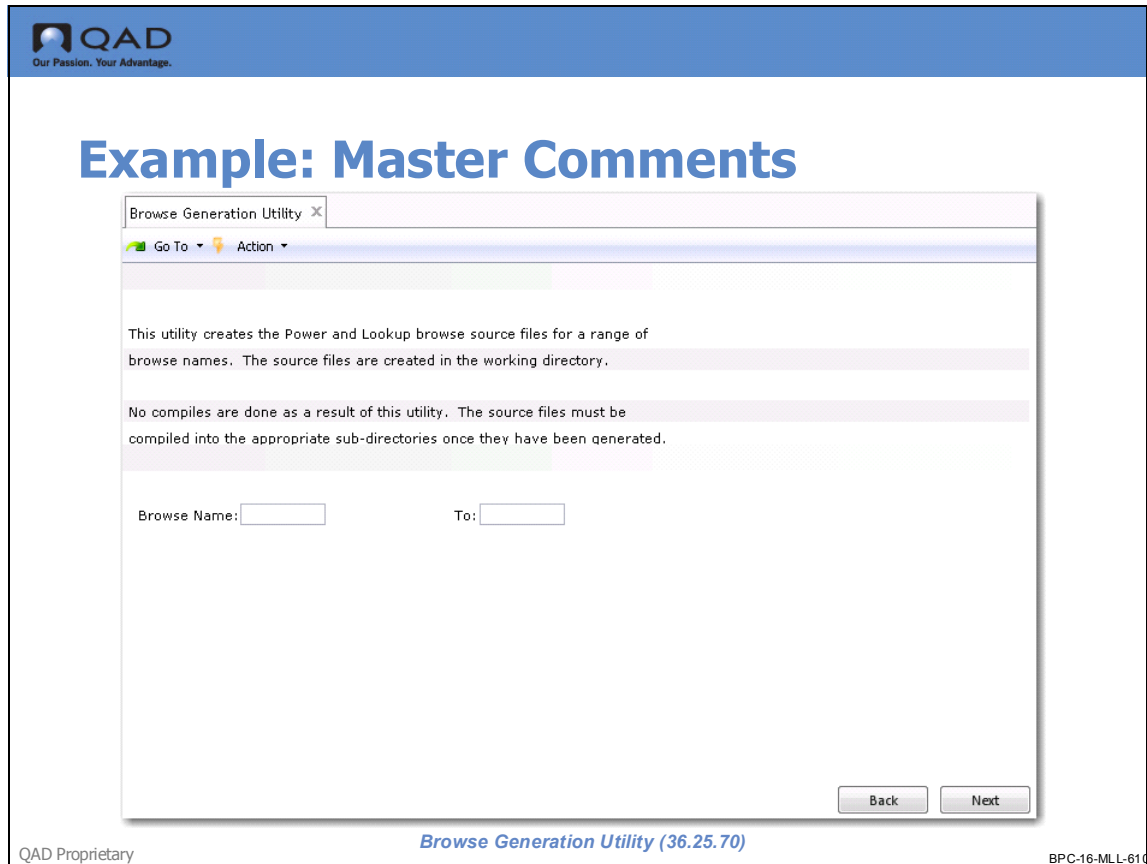
Master Reference: Language:
Type: Page:

This Utility creates the Power and Lookup browse source files for a range of Browse names. The source files are created in the working directory.

No compiles are done as a result of this utility. The source files must be Compiled into the appropriate sub-directories once they have been generated.

gpcmnt.p – Master Comment Maintenance
Use it to enter the text to be displayed in utbrgmt.p

Back Next



Total QAD Enterprise Applications Flow

- ▲ Label Control File
 - Make sure it's YES (if multi-language is required)
- ▲ Language Code Maintenance
 - Make sure that each required language directory is set to the "base" directory
- ▲ User Maintenance
 - Set your userid to the correct language
- ▲ Label Detail Maintenance
 - Link fields to a (existing & translated) term
- ▲ Can be general, but also per program
 - Label Master Maintenance
- ▲ Create and/or translate the term per language



Exceptions

- ▲ DB Validation messages still comes from database
- ▲ yes/no in Message bar

Exercise 5 (25 minutes)

- ▲ Write a simple query to link Item Master, Sales Order and Sales Order Line tables
- ▲ Update your query by using the various label functions explained earlier to make it multi-language
- ▲ Link terms to the various used fields in your query (if required) or create new terms
- ▲ Translate your terms in the various given languages (if required)
- ▲ Test with the various given languages (language per user is set with User Maintenance)

Hands On Exercise 5:

Translation (recap)



Database

so_nbr

SALES_ORDER

SALES_ORDER_LINE

```
&SCOPED-DEFINE sosomt1_p_5 "Sales Order Line"
/* MaxLen: Comment: */
...
define variable l_line as int format ">>9" label {&spsomt1_p_5} ...
form
  so_nbr
  l_line
  ...
with frame a
setFrameLabels(frame a:handle).
...
```

This is where the real
translation takes place

International Multi-Language Labels

- ✓ Characteristics of label making
- ✓ Label Master Terms
- ✓ getTermLabel function and its variations
- ✓ getFrameLabel function
- ✓ Using Message Master
- ✓ Using Master Comments

Appendix A

Programming Standards: Sample Report Template

Use the following template to design your report program

```

{mfdtitle.i “99 “}
{gplabel.i} /* External Label Include -- only add if using mfdeclre.i instead of mfdtitle.i
*/

[Variable definitions...]
[Form definitions...]

/* Set External Labels -- if all fields in the frame use ‘no-label’, then do not need
 setFrameLabels */
 setFrameLabels(frame a:handle).

{wbrp01.i} /* General web report setup */
repeat: /* WITH phrase not allowed here */
[Begin initialize selection criteria]
...
[End initialize selection criteria]
[Begin data entry statement]
if c-application-mode <> 'web' then
[update, set or prompt-for selection criteria...]
with frame a.

{wbrp06.i &command = [update, set or prompt-for]
&fields = “[selection criteria separated by a space] “
&frm = “a” }

[End data entry statement]
[Begin batch quoting for batchable reports and postprocessing of data entry values]
if (c-application-mode <> 'web') or
(c-web-request begins 'data')
then do:

bcdparm = “”. /* if batch can be run */

/* mfquoter.i can be replaced with gpquote.p if the program file size is too large.

```

See gpquote.p header for explanation of parameters */

```

{mfquoter.i [selection criteria]} /* If batch can be run */
“
“

if [invalid selection criteria] then do:
{pxmsg.i ...}
if c-application-mode = 'web' then return.
next-prompt [selection criteria] with frame a.
undo, retry.

end. /* selection criteria validation */
end. /* if data mode or not web */

[End batch quoting for batched reports and post processing of data entry values]
[Assign hi and low criteria ranges...]
[Begin report logic]

```

```

/* See gpseout.i header for explanation of parameters */
{gpseout.i
&printType = "printouts "
&printWidth = 132
&pagedFlag = " "
&stream = " "
&appendToFile = " "
&streamedOutputToFile = " "
&withBatchOption = "yes"
&displayStatementType = 1
&withCancelMessage = "yes"
&pageBottomMargin = 6
&withEmail = "yes"
&withWinprint = "yes"
&defineVariables = "yes"}
{mfphead.i} /* 132 columns */
or
{mfphead2.i} /* 80 columns */
/* Actual report logic may vary from simple for each loop */
for each...
display...
{mfrpch.k.i}
end. /* for each */
{mfrtrail.i} /* 132 columns */
or
{mfrl080.i} /* 80 columns */
[End report logic]
end. /* repeat */
{wbrp04.i &frame-spec = a}

```


Appendix B

Programming Standards: Sample Report Program

Sample Report Program

```

{mfdtitle.i "b+ "}

/* ***** Begin Translatable Strings Definitions ***** */

&SCOPED-DEFINE ppptrp01_p_1 “/no”
/* MaxLen: Comment: */

&SCOPED-DEFINE ppptrp01_p_2 “Lot/Serial!Ref”
/* MaxLen: Comment: */

&SCOPED-DEFINE ppptrp01_p_3 "Include Zero Quantity"
/* MaxLen: Comment: */

&SCOPED-DEFINE ppptrp01_p_5 “Summary/Detail”
/* MaxLen: Comment: */

/* ***** End Translatable Strings Definitions ***** */

define variable abc like pt_abc no-undo.
define variable abc1 like pt_abc no-undo.
define variable part like pt_part no-undo.
define variable part1 like pt_part no-undo.
define variable type like pt_part_type no-undo.
define variable type1 like pt_part_type no-undo.
define variable line like pt_prod_line no-undo.
define variable line1 like pt_prod_line no-undo.
define variable site like in_site no-undo.
define variable site1 like in_site no-undo.
define variable perm like loc_perm format {&ppptrp01_p_1} no-undo.
define variable nettable like is_net format {&ppptrp01_p_1} no-undo.
define variable inc_zero_qty like mfc_logical
    label {&ppptrp01_p_3} no-undo.
define variable ord_tot like in_qty_ord no-undo.
define variable oh_tot like in_qty_oh no-undo.
define variable total_qoh like in_qty_oh no-undo.
define variable found_nonzero as log no-undo.

form
    part colon 15 part1 label {t001.i} colon 49 skip
    line colon 15 line1 label {t001.i} colon 49 skip
    type colon 15 type1 label {t001.i} colon 49 skip
    abc colon 15 abc1 label {t001.i} colon 49 skip
    site colon 15 site1 label {t001.i} colon 49 skip(2)
    inc_zero_qty colon 25 skip
with frame a side-labels width 80.

/* Set external labels */
setFrameLabels(frame a:handle).

{wbrp01.i}
repeat:
    if part1 = hi_char then part1 = “”.

```

```

if line1 = hi_char then line1 = "".
if type1 = hi_char then type1 = "".
if abc1 = hi_char then abc1 = "".
if site1 = hi_char then site1 = "".

if c-application-mode <> 'web':u then
    update
        part
        part1
        line
        line1
        type
        type1
        abc
        abc1
        site
        site1
        inc_zero_qty
    with frame a.

{wbrp06.i &command = update &fields = "part part1 line line1 type
    type1 abc abc1 site site1 inc_zero_qty" &frm = "a"}

if (c-application-mode <> 'web':u) or
(c-web-request begins 'data':u) then do:

    /* Create batch input string */
    bcdparm = "".
        {mfquoter.i part  }
        {mfquoter.i part1 }
        {mfquoter.i line  }
        {mfquoter.i line1 }
        {mfquoter.i type  }
        {mfquoter.i type1 }
        {mfquoter.i abc   }
        {mfquoter.i abc1  }
        {mfquoter.i site  }
        {mfquoter.i site1 }
        {mfquoter.i inc_zero_qty}

    if part1 = "" then part1 = hi_char.
    if line1 = "" then line1 = hi_char.
    if type1 = "" then type1 = hi_char.
    if abc1 = "" then abc1 = hi_char.
    if site1 = "" then site1 = hi_char.

end.

/* Select printer */
{gpselout.i
    &printType = "printer"
    &printWidth = 132

```

```

&pagedFlag = "page"
&stream = " "
&appendToFile = " "
&streamedOutputToTerminal = " "
&withBatchOption = "yes"
&displayStatementType = 1
&withCancelMessage = "yes"
&pageBottomMargin = 6
&withEmail = "no"
&withWinprint = "yes"
&defineVariables = "yes"}

```

```
{mfphead.i}
```

form with frame b down width 132 no-attr-space.

```
/* SET EXTERNAL LABELS */
```

```
setFrameLabels(frame b:handle).
```

```
for each in_mstr
```

```
    fields(in_abc in_cnt_date in_part in_qty_oh
```

```
           in_qty_ord in_site)
```

```
no-lock
```

```
where (in_part >= part and in_part <= part1) and
```

```
       (in_site >= site and in_site <= site1) and
```

```
       (in_abc >= abc and in_abc <= abc1)
```

```
break by in_part by in_site:
```

```
for first pt_mstr
```

```
    fields(pt_part pt_part_type
```

```
           pt_prod_line pt_um)
```

```
no-lock
```

```
where pt_part = in_part
```

```
    and pt_prod_line >= line
```

```
    and pt_prod_line <= line1
```

```
    and pt_part_type >= type
```

```
    and pt_part_type <= type1:
```

```
end. /* FOR FIRST pt_mstr */
```

```
if not available pt_mstr then next.
```

```
assign
```

```
    total_qoh = 0
```

```
    found_nonzero = no.
```

```
for each ld_det
```

```
    fields(ld_loc ld_lot ld_part ld_qty_oh
```

```
           ld_ref ld_site ld_status) no-lock
```

```
where ld_part= pt_part and ld_site = in_site
```

```
    and ld_qty_oh <> 0:
```

```

        assign
            total_qoh = total_qoh + ld_qty_oh
            found_nonzero = yes.
    end. /* FOR EACH lad_det */

    if inc_zero_qty or found_nonzero then do:
        display
            pt_part format "x(26)"
            pt_um
            in_abc
            in_site
            in_cnt_date
            in_qty_ord
            total_qoh @ in_qty_oh format "->>>>>>>>>9.9<<<<<<<<<<"
        with frame b.

        down 1 with frame b.

    end. /* IF inc_zero_qty OR found_nonzero THEN DO */
    {mfrpch.k.i}

    end. /* for each in_mstr ... */
    {mfrtrail.i}

end. /* repeat: */

{wbrp04.i &frame-spec = a}

```


Appendix C

How to Web-Enable a Report

1 Introduction

This document explains the fundamentals of how standard QAD Enterprise Applications reports are made to function in a Web environment.

A report run in a web environment is run in two phases. The first phase gets the report's input variables and their related frame to display in the Java User Interface. In the second phase, the Java UI passes the report's input variables and their values into the report program. Then runs the report to generate the output. On completion of the second phase, the report's output is sent to the Java UI.

There are two slightly different methods used to web-enable a report. Both methods share the same include files but use them slightly differently. Using these include files most QAD Enterprise Applications standard reports can be web-enabled.

2 How Web-Enabled Reports Function

In the normal QAD Enterprise Applications environment a report is run only once and runs from start to finish. In the Web environment a report is run twice. The first run outputs the report's input variables and their associated frame to the Java User Interface. The second run populates the report's input variables and generates the report's output for the Java User Interface.

To efficiently complete both runs a web-enabled report examines parts of the input "query string" that are passed from the Java environment. There are two basic types of "query strings": The "INIT" query string that causes the output of the report's input frame and variables; And the "DATA" query string that causes the report to generate its output. On the initial run, the report returns to its calling program without stepping into the body of the report. On the data run, the report skips the output of the input frame and goes straight to accepting the values of its input variables. This control over the report's logic is handled by the web include files and other web enabling code inserted directly into the Figure 1 shows a query string for the initial run of report.p report.

```
MAINAPP=RUN&REQUEST=INIT&APP=report.p
```

Figure 1

Figure 2 shows a generic data query string for a report.

```
MAINAPP=RUN&REQUEST=DATA
&APP=report.p
&DATA=field%12value%11field%12value%11..
```

Figure 2

3 Web Report Include files

Table 1 describes the four include files used to web-enable a report.

Wbrp01.i	&io-frame**	Placed in the main report program after the form definitions and before the outermost iterating loop in the report. This include: Sets-up necessary persistent procedures; checks security access rules; and formats the web output stream. The report's frame to update input variables is accepted in parameter &io-frame.
Wbrp06.i	&command &fields &frm ***	Placed near the "update", "set", or "prompt-for" statement that populates the report parameters. On an "INIT" request wbrp06.i outputs the screen to the Java User Interface. On a "DATA" request wbrp06.i updates the report's input variables and redirects the report's output in the web stream. The report's command to populate its input variables, either "update", "set", or "prompt-for", is required by wbrp06.i in the parameter &commands. The report's input variables are required by wbrp06.i in the parameter &fields. The report's frame to update its input variables is accepted in parameter &frm.
Wbrp02.i	&io-frame**	Placed after the form definitions and before the outermost iterating loop of web-enabled report sub-programs. It defines shared variables and shared handles so that the sub-report can be web-enabled. The report's frame to update input variables is accepted in parameter &io-frame.
Wbrp04.i*	&frame-spec	Placed at the bottom of the file it contains internal procedures that can be called by wbrp01.i and wbrp02.i. Note that, wbrp04.i is not required if the parameter io-frame is specified, in wbrp02.i or wbrp01.i, in the program. The report's frame to update input variables is accepted in input parameter &frame.

*	Include wbrp04.i is not required if the parameter io-frame is specified, in wbrp02.i or wbrp01.i, in the program.
**	Optional Parameter.
***	The "&frame" parameter in wbrp06.i is not required if the parameter io-frame is specified, in wbrp02.i or wbrp01.i, in the program.

The option to include &io-frame, &frm, and wbrp04.i depends on which method of web-enabling a report is chosen. The standard method would look similar Figure 3.

```

{wbrp01.i} or {wbrp02.i}

if c-application-mode <> 'WEB':U then
  update var1 var2 var3 with frame a.

{wbrp06.i &command = update
  &fields = ' var1 var2 var3 '
  &frm = 'a'}

{wbrp04.i &frame-spec = a}

```

Figure 3

Note that in the standard method &io-frame is not specified while &frm is specified and wbrp04.i is included. In the modified method the code would look more like Figure 4.

```

{wbrp01.i &io-frame = a} or {wbrp02.i io-frame = a}

if c-application-mode <> 'WEB':U then
  update var1 var2 var3 with frame a.

{wbrp06.i &command = update
  &fields = ' var1 var2 var3 '}

```

Figure 4

Note that in the modified method &io-frame is specified while &frm is not specified and wbrp04.i is not included.

4 Conversion Exceptions and Conditions

There are some report programs that will require more work to web-enable because of the current program structure. Some of the conditions to watch for are; pause or message statements, shared frames, more than one statement to populate input variables, or subprogram that require web functionality.

a “Message” Statements:

A “message” statements should not be executed in web mode. Therefore the statement should, at minimum, be handled as in Figure 5.

```

If c-application-mode <> 'WEB':U then
  Message.

```

Figure 5

b New Shared Frames

New shared frames cannot exist in the first report program. This is because the first report program is run persistently within the web environment. Therefore, any new shared frame definition in the first report program will need to be moved to a subprogram.

This can be accomplished in two ways. The first way is to move the shared frame definition and all references to the frame into a subprogram called by the report. Or secondly, make the first report program a wrapper for a new subprogram and move all functionality into the new subprogram.

There are code examples for dealing with shared frames in section 5 Common Report Formats.

c More Than One Statement to Populate Report Input Variables.

If an “update”, “set”, or “prompt-for” to populate report input variables exists in more than one program then the report cannot be converted at this time.

If two or more “update”, “set”, or “prompt-for” statements exist in one program it maybe possible to web-enable the report. It is possible to web-enable the report if the second statement is not conditional on the first statement.

Figure 6 shows how to convert the most common case of a “prompt-for” followed by an “update” statement.

```
{wbrp06.i &command = prompt-for
  &fields = 'var1 var2 var3 var4 '
  &frm   = 'a'}

/* update input variable only on second web run */
if c-application-mode = 'WEB':U and
  c-web-request begins 'data' then
  assign var2
    var3
    var4.

if c-application-mode <> 'WEB':U then
  prompt-for var1
  with frame a.

/* validation for var1 */
if cannot find record xxx
  where xxx.field1 = input var1 then do:
  display a message.
  undo and retry the outer loop.
end.

if c-application-mode <> 'WEB':U then
  update var2
    var3
    var4
  with frame a.
```

Figure 6

First, notice that “&command = prompt-for” this is because if the command is “prompt-for” then wbrp06.i doesn’t assign the screen-values of &fields to the field-values of &fields. Next notice the “assign” statement assigns the screen-values of var2, var3, and var4 to their field-values. Var1’s screen-value is not assigned to its field-value since a “prompt-for” statement doesn’t update the field-value for a variable. Note: this is using the standard method since &frm is specified in wbrp06.i.

d Subprograms

All top-level report programs must be enabled web-enabled regardless of the logic in the report program. This is because the report’s parent program sets a handle pointing to the report.

Contrary to top-level report programs only some subprograms need to be web-enabled. Generally, a subprogram needs to be web-enabled when it contains web functionality beyond checking the variable “c-application-mode”. One method of knowing if a program contains web functionality is to look for wbgp03.i in the subprogram or its include files. The presence of wbgp03.i indicates web functionality and the program will need to be converted.

If a subprogram contains web functionality it will need the include wbrp02.i to define shared variables and handles. If the subprogram contains the “update”, “set”, or “prompt-for” statement for populating the reports input variables then the subprogram will need the include wbrp06.i.

Currently, the most commonly used include files that have web functionality are listed in Table 2.

wfselbat.i	mfselbpr.i	mfselp01.i	mfselp02.i
mfselp03.i	mfselp04.i	mfphead.i	mfphead2.i
mfrpexit.i	mfrtrail.i	mfreset.i	mfrl080.i
mfreset2.i			

Table 2

e Web block around Report Variable Validation

In many of the reports the Figure 7 code can be seen:

```
{wbrp06.i &command = update
  &fields = 'var1 var2 var3 '
  &frm = 'a'}

if (c-application-mode <> 'WEB':U) or
(c-application-mode = 'WEB':U and
(c-web-request begins 'DATA':U)) then do:

  input variable validations.

end.
```

Figure 7

This code is currently unnecessary and therefore it is not explained in great detail in this document.

5 Common Report Formats

Here are some examples of common report layouts in standard and modified formats.

a The Simple Reports:

Simple Report with the Standard Format

```
{mfdtitle.i "99 "}

variable definitions.

form definitions.

{wbrp01.i}

mainblock:
repeat:

  if c-application-mode <> 'WEB':u then
    update var1 var2 var3
    with frame a.

  {wbrp06.i &command = update
    &fields = ' var1 var2 var3 '
    &frm   = 'a'}

variable validations.

{mfselbpr.i "printer" 132}

report logic.

{mfrtrail.i}

end. /* mainblock */

{wbrp04.i &frame-spec = a}
```

Simple Report with the Modified Format

```

{mfdtitle.i "99 "}

variable definitions.

form definitions.

{wbrp01.i &io-frame = a}

mainblock:
repeat:

  if c-application-mode <> 'WEB':U then
    update var1 var2 var3
    with frame a.

  {wbrp06.i &command = update
    &fields = ' var1 var2 var3 '}

variable validations.

{mfslbpr.i "printer" 132}

report logic.

{mftrail.i}

end. /* mainblock */

```

Simple Report with Subprogram (Modified Format)**Report.p**

```

{mfdtitle.i "99 "}

variable definitions.

form definitions.

{wbrp01.i &io-frame = a}

mainblock:
repeat:

  if c-application-mode <> 'WEB':U then
    update var1 var2 var3
    with frame a.

  {wbrp06.i &command = update
    &fields = 'var1 var2 var3'}

variable validation logic.

{mfslbpr.i "printer" 132}

report logic.
{gprun.i "report2.p"}

{mfrtrail.i}

end. /* mainblock */

```

Report2.p

```

For each xxxxx no-lock:
  Dislay xxxxx.field1
  xxxxx.field2
  with frame b.
end.

```

b Report with a Web-Enabled Subprogram:

Since report2.p contains mfrpch.k.i and mfrpch.k.i includes wbgp03.i then report2.p must contain wbrp02.i.

Standard Format**Report.p**

```
{mfdtitle.i "99 "}
```

variable definitions.

form definitions.

```
{wbrp01.i}
```

mainblock:

repeat:

```
{gprun.i ‘report2.p’}
```

```
{mfselbpr.i "printer" 132}
```

For each xxxxx no-lock:

 Dislay xxxxx.field1

 xxxxx.field2

 with frame b.

```
/* include containing web functionality */
```

```
end. /* for each xxxxx */
```

```
{mfrtrail.i}
```

```
end. /* mainblock */
```

```
{wbrp04.i &frame-spec = a}
```


Report2.p

variable definitions.

form definitions.

```
/* contains web functionality */
{wbrp02.i}
```

```
if c-application-mode <> 'WEB':U then
  update var1 var2 var3
  with frame a.
```

```
{wbrp06.i &command = update
  &fields = 'var1 var2 var3'
  &frm    = 'a'}
```

variable validation logic.

c Reports with a 'New Shared frame':

Create Wrapper and New Subprogram

Report.p

```
/* moved all logic to the sub program */
```

```
{mfdtitle.i "99 "}
{wbrp01.i}
```

```
{gprun.i "report2.p"}
/**** comment out rest of program ...
```

Report2.p

```
{mfdtitle.i "99 "}
```

variable definitions.

form definitions.

```
/* note even though this is technically a sub- */
/* program its treated like a top level program. */
{wbrp02.i &io-frame= a}
```

mainblock:

repeat:

```
if c-application-mode <> 'WEB':U then
  update var1 var2 var3
  with frame a.
```

```
{wbrp06.i &command = update
  &fields = ' var1 var2 var3 '}
```

variable validation logic.

```
{mfselbpr.i "printer" 132}
```

report logic.

run subprograms.

```
{mfitrail.i}
```

```
end. /* mainblock */
```

Extract Shared Frame Logic Approach (Standard Format)**Report.p**

```
{mfdtitle.i "99 "}
```

variable definitions.

form definitions (**with reference to new shared frame removed**).

```
{wbrp01.i}
```

mainblock:

repeat:

```
if c-application-mode <> 'WEB':U then
  update var1 var2 var3
  with frame a.
```

```
{wbrp06.i &command = update
  &fields = 'var1 var2 var3'
  &frm   = 'a'}
```

variable validation logic.

```
{mfselbpr.i "printer" 132}
```

report logic.

```
/* moved shared frame logic to newsubprog.p */
gprun.i 'newsubprog.p'}
```

```
{mfrtrail.i}
```

end. /* mainblock */

```
{wbrp04.i &frame-spec = a}
```

newsubprog.p

variable definitions.

form definitions.

/* include for new shared frame */

{nsframe.i “new shared”}

{wbrp02.i}

/* shared frame logic */

For each xxxxx no-lock:

display xxxxx.field1

xxxxx.field2

with frame b.

{gprun.i ‘report2.p’}

{mfrpch.k.i}

end. /* for each xxxxx */

Report2.p

variable definitions.

form definitions.

/* include for shared frame */

{nsframe.i “shared”}

For each yyyyy no-lock:

Dislay yyyyy.field1

yyyyy.field2

with frame b.

end. /* for each yyyyy */

Appendix D

GUI Converter

The GUI converter is used to convert a prepared CUI (character) program into a GUI (graphical) program. The total conversion process consists of the following steps:

- Add a ConvertMode-line to all programs
- Scan programs for possible conversion problems
- Convert the sources
- Examine the convert logging
- Compile the sources
- Test the sources

This is an iterative process, the last steps will probably be performed more than once. To be able to perform the automatic conversion, you have to follow many programming standards. In this chapter these standards will be discussed. Furthermore several recommendations are made to speed up the conversion process.

1 Add a ConvertMode-line to all Programs

The goal is to make the source for QAD Enterprise Applications version 7.3 and higher GUI enabled. This means it can pass through the source converter to produce source code that compiles and runs correctly under GUI version of PROGRESS 7.3 and above. Thus producing QAD Enterprise Applications 8.X versions.

The first step is to add a “V8:ConvertMode” line to all your programs.

```
/*V8:ConvertMode=Maintenance */
```

This directive tells the Converter to treat the source file as a maintenance program. This means that no output is expected, all frames will have attributes appropriate for screen display, and there could be several levels of nested transactions.

```
/*V8:ConvertMode=Report */
```

This directive tells the Converter that the program being converted is a report program that doesn't quite fit the standard model. Programs converted as Report function the same as FullGUIReport, but don't appear as “GUI” on the screen.

```
/*V8:ConvertMode=FullGUIReport */
```

This directive tells the Converter to treat the source file as a report program. This means that there will be one frame, named frame a, that accepts user input and will have screen display attributes, and all other frames will be used for hardcopy-style output. Only one REPEAT block is expected to be used for generating the output.

The output source program to be generated will fit with the event-driven user interface model and look more “GUI.”

```
/*V8:ConvertMode=NoConvert */
```

This is another type of file that the Converter recognizes. When this ConvertMode= type is found, the input file is copied to the target directory with a system “copy” command.

Less often used are:

```
/*V8:ConvertMode=ReportAndMaintenance */
```

This directive tells the Converter that the program being converted is to have both Report and Maintenance attributes. The appropriate attributes are activated at compile time based on pre-processor definitions inserted by the converter.

Example file : mfsotr1a.i
 Example usage : /*V8:ConvertMode=ReportAndMaintenance */
 /*V8:ConvertMode=NoConvertNoOutput */

This is another type of file that the Converter recognizes. When this ConvertMode= type is found, no output is produced. This does not cause an error condition.

/*V8:ConvertMode=ConditionalIncludeExclude */

This is another type of file that the Converter recognizes. When this ConvertMode= type is found, the only processing done to produce the output file is to move mfdeclre/mfdtitle and process the /*V8! ... */ and /*V8-*/ ... /*V8+*/ directives.

Note These convert-mode lines must be used verbatim (that is, exactly as typed in the examples above).

2 Scan programs for possible conversion problems

To support single-source between QAD Enterprise Applications 7 and QAD Enterprise Applications 8, a feature has been built into the QAD Enterprise Applications GUI Source-Converter that allows source to be conditionally included into QAD Enterprise Applications source code based upon whether the environment is QAD Enterprise Applications V7 or V8. Code can be 'injected' into a file for QAD Enterprise Applications Version 8 by putting it into a special comment that will turn into active code when it passes through the converter. The code can be 'cut out' of a source file by surrounding it with special leading and trailing comments that turn into one big comment when passed through the converter.

There are several tokens that can be embedded in the program to help the Converter produce the desired output. These tokens are listed below. They must be used verbatim!

/*V8! ... */

This is used to enable code only for QAD Enterprise Applications v8.x. The Converter removes the leading and trailing comments during the prepass and then converts the previously enclosed code during the output-generating pass.

Before Conversion: message "Whoa!" /*V8! view-as alert-box */.

After Conversion: message "Whoa!" view-as alert-box.

/*V8-*/ ... /*V8+*/

This is used to strip code out of the Converted version of the file. Code found between these tokens is not processed by the Converter and not written to the output file.

Before Conversion: message "Smell the roses." /*V8-*/ pause. /*V8+*/

After Conversion: message "Smell the roses." /*V8+*/

Less often used are:

/*V8:HiddenDownFrame=<frame name> */

Down frames require the BGCOLOR 8 attribute to be added to them. This directive is used to help the converter identify frames as down frames. If the frame is defined in an include file, this directive helps the converter generate the correct code.

Example file: sosomta.p

Example usage: /*V8:HiddenDownFrame=c */

/*V8:RPMainCaller=<frame name> */

The Converter doesn't process nested include files during parsing. This directive is needed for Report programs where most of the code recognized as Report code is in an include file.

Example file: ?

Example usage: `/*V8:RPMainCaller=frmnam */`

`/*V8:EditableDownFrame=<frame name> */`

Down frames require the BGCOLOR 8 attribute. The exception is that some down frames are used for input. These down frames need the 3D attribute to look consistent with the rest of QAD Enterprise Applications editable fields. This directive is used to identify a down frame as being editable.

Example file: gracmta.p

Example usage: `/*V8:EditableDownFrame=c */`

`/*V8:DontRefreshTitle=<frame name> */`

Because the frame title is added by the Converter and isn't really a standard frame title, it needs to be refreshed after each "CLEAR." Some frames titles are built with values only known at title creation time and can't be reconstructed. This directive tells the Converter to not refresh the title of a particular frame.

Example file: fssamta.p

Example usage: `/*V8:DontRefreshTitle=c */`

`/*V8:NonStreamIoFrame=<frame name> */`

Some Report programs don't fit the "Report" program model and have frames other than frame a that should not get the STREAM-IO attribute. This directive identifies those frames and allows the converter to generate appropriate output.

Example file: csptiq3a.p

Example usage: `/*V8:NonStreamIoFrame=c */`

`/*V8:ConcatString */`

This directive is used in text files. Progress v6 could handle one string of text per line, Progress v7 requires one long, multi-line text string. This directive puts the Converter in a special mode that will only handle this type of files.

Example file: t029.i

Example usage: `/*V8:ConcatString */`

3 Rules to avoid problems

Described here are coding rules to avoid conversion problems. First the recommendations and standards are discussed, which were discovered during the conversion of MLP and TIPAC. Furthermore additional standards can be found.

Specific rules

Use the same converter version as QAD did.

QAD has distributed several releases of QAD Enterprise Applications 7.3K, which differ in the way they were prepared for conversion to 8.3K. These changes often had to do with changes made to the convert program. Because of this you should basically decide to use the same converter version as QAD did. One of the problems that occurred because of this is stated below.

We kept on having runtime shared frame problems between the TIPAC source ppptmta.p and the standard QAD source ppptmtd.p. Finally it turned out that the standard 8.3K sources were converted from 7.3K using version 1.60 of the GUI converter, while we were using version 1.66 to convert the TIPAC sources. These versions converted the FORM statements of shared frames (in this particular case frame-name d0) slightly different.

We solved the problem by converting all TIPAC sources again, now using the GUI converter version 1.60.

Sequence: modify 7.3K sources, convert and use them under GUI 8.3K.

It's important to keep modifying 7.3K sources and run them through the GUI converter in order to get good 8.3K sources. This way we only have to support the GUI prepared 7.3K sources.

The GUI converter is not capable to convert a FORM statement that names a record (especially mfc_ctrl).

The converter places skips in front of and after the record name. This works fine if a FORM statement summarizes field names, but you'll get compile errors when using the record name. Both TIPAC and MLP had a control file program with this problem. You can solve it by naming all the fields of the record in the FORM statement. There is a Progress Dictionary Tool that creates an ASSIGN statement for a record. This could be helpful for records with many fields.

Converter version 1.60 has been designed for 7.3K. Version 1.69 has been designed for 8.5D.

All TIPAC programs have been converted using GUI converter version 1.60, except for tptisoq.p and tptiiq.p. Those programs gave shared frame problems. For these two we have used version 1.66 because it supports an additional ConvertMode:

ReportANDMaintenance! We've used that ConvertMode because the two sources are a mixture of Maintenance, Inquiry and Report.

Do not load a dump file into the GUI database more than once.

In QAD Enterprise Applications 8.3K you can add field help with menu option 36.20.1 or by typing insert drl_mstr in the editor. The file drl_mstr is part of the GUI database. Be careful when loading .d-files in this database. Loading the same .d-file more than once can cause duplicate records while some files only have non-unique indexes!

The standard GUI converter program is not able to convert WHEN options in UPDATE statements for sources with ConvertMode FullGUIReport.

Ask the 'product management' for a adjusted version of converter 1.69, or change the logic for updating the selections, leaving out all WHEN options. You can try to add an editing phrase that conditionally skips fields. In the MLP program mlpworp.p we've just commented out all WHEN options. This is probably not acceptable because by doing this, the functionality changed as well.

Add a space at the start and end of the longest line in the frame, to make sure that labels do not overwrite or touch the frame borders and always use explicit positioning within a frame.

When a frame has a free-floating field on the left frame border (no COLON, TO, or AT statement), always precede it with a space(1), or use an AT 2 to locate the field. When a frame has no explicit WIDTH (e.g., small overlay frames), always add a SPACE(2) after the last field on every line. This leaves enough space on the right side of the frame for the V8 decoration rectangle.

When aligning text vertically on a form, DO NOT use hard-coded spaces. The spaces are a different size than other characters in a proportional font world. Use the AT, TO, and COLON keywords to start or end Text in specific columns;

Correct : "Note:" at 2

Incorrect : "Note:"

When aligning both labelled and unlabeled fields on a form, use COLONS for the labelled and ATs for the unlabeled fields.

Always use an explicit WIDTH 80 for on-screen Data-entry frames that are supposed to be full-screen-width; do NOT rely on a field pulling the frame to column 80 such as the "output" field on a report's main frame. In short, you should always put a WIDTH on every frame except overlay frames.

General directives

Normally standard QAD Enterprise Applications doesn't use titles above Inquiry frames.

Some TIPAC programs, like tptiiq04.p use titles above Inquiry frames. In 8.3K the title is not centred anymore, and there is a small cross on the right side of the title bar as if you could terminate the frame. However, the small cross is never enabled. We've tried many things, but we didn't get it to work.

Most Inquiries are replaced by Browsers, even in the 8.3K version.

If the output in a Help Browser looks strange, first try if you can move using PgUp, and then check the default values in the Filter. Dates for example, sometimes default to the system date (TODAY).

One of the arguments of the convert program is -ini.

It can be used to supply a different name for the convert.ini file. This way it's possible to make separate ones for TIPAC, MLP, TOOL, etc.

The converter generates grey boxes for Maintenance frames and white ones for Report and Inquiry frames.

Inquiry and Report programs can consist of 1 frame to handle input, the name has to be 'a', and only 1 update statement.

The converter cannot handle more than 1 update statement. If you run into serious problems, you can decide to split up your program. Move the Inquiry or Report part to a separate program (ConvertMode can be Report or FullGUIReport). This part can even have it's own frame a with 1 update statement. The ConvertMode of the remaining program has to be changed to Maintenance.

Some programs have a one-line update frame at the bottom of the screen and a down frame right above it to hold a few of the updated lines.

To make sure the editable frame looks the same as the down frame, we have to provide the following directive for the down frame `/*V8:EditableDownFrame= <frame-name>*/`. Our advice is to put this directive just in front of the FORM statement. The down frame turns grey and wider, looking exactly like an editable frame.

A source file must be strictly a non-printing program/subprogram (for example, Maintenance/data-entry), or a printing program/subprogram (for example, Report/Inquiry/Print) with a single input frame named “frame a.”

There shall be no mixing of output capabilities with data entry programs. For example, if a maintenance program has a “Print Audit Trail Report” capability, then put all the output commands including the standard printer include files (mfselprt.i etc.) into a separate subprogram file and run it from the maintenance program. See `dmaprv.p` & `dmaprvp.p` for example.

Do not use any of the new PROGRESS 7 keywords even when programming in PROGRESS V6.

The common current violations are “forward, back, process and window.”

Remove unused “view frame XXX” statements for frames that no longer exist in the scope of the program (mostly due to splitting of code into another .p-files).

Always use explicit SKIPS at the end of frame lines whenever the COLON or AT alignment won't cause the SKIP naturally.

Do not rely on field fit to cause SKIPS (GUI can often fit more on a line so small fields will move up a line skewing everything badly).

All Fields on an input form must be defined within a single FORM statement.

Do not put fields on forms via SET, UPDATE, PROMPT-FOR, or DISPLAY statements, or in a secondary FORM statement for the same frame name. The only exception is the “Output” and “Batch ID” fields added to FRAME A by the `mfselbpr.i` and `mfselprt.i` include files.

Define all data input frames in a FORM statement.

Do not define frames ad-hoc on data-entry statements (for example, Update/Set/ Prompt-for .. with frame X...). The converter needs to modify the form definitions and can only do so on FORM statements.

Batchable printing programs (reports) must only have one input frame named “FRAME A,” and only one SET, UPDATE, or PROMPT-FOR statement.

Non-batchable printing programs (inquires and prints) can have auxiliary pop-up overlay frames for input, and can have multiple data-entry statements. It is preferable to NOT use CHOOSE logic for input frames in printing programs.

The “mfdtitle.i” include must only occur in menu-level programs, not in subprograms, AND should never occur twice in a program-calling sequence.

NEVER place frames above line 3 !!!

Don't use explicit ROW placement in frame-phrases except with overlay frames.

Make them relative to other frames whenever possible, not hard coded rows.

Using WHEN.

Whenever a WHEN clause is used in a Data-entry statement, enclose the conditional expression within parenthesis. For instance,

```
update l_one l_two when (cnc_logical) l_three.
```

Always use the “scroll” option when defining a CHOOSE frame, even if it is not needed because the frame can fit all lines. “SCROLL 1” should always be safe. This is a key phrase the converter needs to see to size and color the frame correctly.

Using validate.

Currently embedding “validate()” functions within Data-entry statements, causes problems with conversion. However, this coding practice is needed for sharing frames. If you are not sharing frames, or can get around the use in another way, please avoid putting validate()'s in Data-entry statements. They can be placed in the FORM statements.

4 Convert the CUI sources into GUI sources

a Create a convert.ini file

```
srcSearchPath=/ext5/mfg83kdu/minkels/werk
targetDir=/ext15/convert/minkels
workDir=
```

An initialisation file is required to control part of the Converter behavior. Below are the contents of a sample file.

srcSearchPath is the location of the CUI-sources. Source files will be read from the directories in the comma delimited search path. It must have at least one valid directory. targetDir is the location of the converted sources. Output files will be written to “targetDir=”. The output filename is the same as the input file. A valid, non-blank target directory specification is required.

workDir is the current directory. Listfiles will be read from “workDir=”. The work directory defaults to the local directory if left blank.

b Create a convert-script

Below is an example of a UNIX script, but these same principles can be used for Windows '95 or NT;

```
rm conv_minkels.log
for i in /ext5/minkels/mfg83kdu/werk/source/*.piw]
do
    conv166/convert -ini=conv_minkels.ini $i >> conv_minkels.log
done
```

Logging messages will be written into the ‘conv_minkels.log’-file.

‘conv166/convert is the actual converter. It is located in the conv166 directory. The converter version is indicated by the directory name. Three versions are available.

Version 160

Used to convert QAD Enterprise Applications 7.3K sources into 8.3K sources. Some programs will cause problems while testing in GUI. Errors like ‘shared frame x has not yet been created’ will appear. The standard equivalents of those programs could be converted with version 166. The customized programs should therefore also be converted with version 166.

Version 166

Not used very often, basically an enhanced example of 160.

Version 169

Used to convert QAD Enterprise Applications 7.5D sources into 8.5D sources. One known shortcoming of this example is the conversion of several warehouse programs. Our colleagues of QAD UK have made their own version of 169 in order to compile those warehouse sources correctly.

c Execute the converter

Start the converter by typing:

```
./<convert-script> [ENTER]
```

After the input file, output logfile and output source file has been opened, the processing can begin. A prepass is made through the file to help determine how it should be processed. Next the file is traversed again, and the output file is generated based on the input file. Certain constructs and features in the input file trigger how the converter generates output.

PreProcessing Pass

A pass is made through the file before any output is generated. These steps are done during the pre-processing pass:

- 1** The existence of mfdtitle.i, mfdeclre.i, mfselptr.i, mfselbpr.i is checked to help determine the file type.
- 2** Processing block statements are parsed to determine the “on error” actions. These actions are stored for possible future use during output generation.
- 3** If not specified on the command line, the convert mode is determined from the ConvertMode= directive.
- 4** Converter conditional code is enabled or removed based on the /*V8! */ and /*V8-*/ /*V8+*/ constructs.

At the end of the pre-processing loop, if the convert mode was NoConvert, the original file is copied to the target directory, and processing for this file is terminated. If the convert mode is unknown an error message is generated.

Processing Pass

Another pass is made through the file for the purpose of generating output. During this pass, the input file is copied to the target directory a line at a time. Upon encountering certain Progress keywords and other constructs, the convert will add to, remove from, or modify the code as it is being sent to the target directory.

File ID

The first line written to the output file is a comment containing file creation information. For example:

```
/* GUI CONVERTED from cncfor.i (converter v1.60) Wed Sep 25 15:14:41 1996 */
```

FORM Statements

The converter modifies all form statements. Report frames need to be told to use only fixed size fonts during output. Maintenance frames use proportionally spaced fonts. Both sets of attributes are applied to each frame, and the correct ones are chosen at compile time.

The variable *global-beam-me-up*

At any point during Maintenance program data entry, the user can select Close from the menu accessed by clicking on the icon at the left end of a window's top border. This action needs to be checked so control can be returned to the previous window. This functionality is new to QAD Enterprise Applications, and is standard in the Windows environment. The converter adds at every applicable transaction level the code for setting and checking a global variable, global-beam-me-up.

Full GUI Reports

Programs converted as FullGUIReport have more of a “Windows” look. This involves more code manipulation than a Report conversion, and requires input programs to have a more consistent, predictable structure.

FullGUIReport input programs are output as several procedures. The procedures handle the display and updating of the report selection criteria, as well as quoting the bcdparm values and generating the report output. For more information on procedures, consult the Progress v7 documentation. Common include files, such as mfrpexit.i, are replaced with Progress v7 equivalents.

5 Examine the converted Output

The Converter generates two kinds of output, converted source and messages.

The interesting output of the conversion process is a Progress v7 file. This file is based on the input source code and has been modified to conform to QAD Enterprise Applications GUI standards. In theory, only the look and feel of the input source program will have been changed to produce the output file; there is generally no change in functionality.

The output file created will have the same name as the input file, but will be written to the directory specified in the initialisation file. The output file will overwrite any existing file with the same name.

Messages are described in detail in a separate section of this document. Messages include information, warnings, and errors encountered during the conversion of a Progress V6 file and the creation of the Progress v7 file. When converting files from a Listfile, messages are written to a log file. The log file name is specified in the convert file described in paragraph 0. When the command line specifies the name of the input source file(s), the messages are written to stdout.

There are three types of messages that can be generated by the converter, what causes the message, and, if applicable, what can be done to avoid the message in the next conversion attempt.

Informational messages tell that something unusual or suspicious has occurred during conversion.

Warning messages are generated when a condition in the input source file requires careful verification of the output.

Error messages cause the converter to quit processing the current file. These are generally violations of the GUI coding standards described in another document.

Only the word QUIT (in capitals) indicates a failure. See the appendix for an example logging file and a description of all possible messages. If the programs closely follow QAD standards then you should have few failures.

6 Compile the GUI sources

Before you are able to compile, make sure the GUI environment is working properly on your P.C. Setting up the GUI environment is strictly beyond the scope of this document.

Start QAD Enterprise Applications 8.x by double-clicking on 'mfgontw'. The first screen is a picture of QAD Enterprise Applications with a globe. If the text exceeds the border of the picture, you need to adjust your display settings again.

Then open the progress-editor by clicking on 'the menu-bar and open the file 'compillex.w' and run the program. The compilation will then start OR one can use the QAD Enterprise Applications customized software compiler (which starts directly the compillex.w program).

The compiling program will log all errors and warnings into log-file 'utcompil.log'. ; Possible warnings/errors can be:

Warning :	**Cannot fit FILL-IN with AT within FRAME f. (4027) Ignoring position info for AT (2054)
Problem :	Adjusting the display settings (VGA mode + small fonts)
Warning :	Could not find rect-frame-label in frame a. (3567)
Problem :	Conflicting ConvertMode
Warning :	** Unknown Field or Variable name - frame-report-cancel. (201)
Problem :	Conflicting ConvertMode
Warning :	** Phrase or option conflicts with previous phrase or option. (277)
Problem :	Conflicting ConvertMode
Warning :	widget-attribute or reserved keyword. (405)
Problem :	A definition with the same name as a progress keyword.
Warning :	Action Segment has exceeded its limit of 61440 bytes (3307)
Problem :	An object is divided into segments, e.g. all variable definitions. The size of a segment is influenced by the use of internal procedures.
Warning :	**LITERAL (will not fit in FRAME in (4028)
Problem :	Starting position, width of frame or string is too long.

7 (Flow) Test the sources

The converted sources should be checked on their appearance. The functionality remains unchanged, but the most common problems related with appearance are progress cmessages and unexpected lay-out changes

Warning :	Invalid widget-handle. Not initialized or points to a deleted widget. (3135)
Problem :	An invalid widget-handle has been encountered. This typically occurs when querying or setting a widget attribute.
Warning :	Cannot access the <attribute name> attribute because the widget does not exist. (3140)

Problem :	The widget handle used when setting a widget's attribute was not valid. This is typically caused by using a widget handle that has not been initialized, or the widget that the widget-handle was pointing to was destroyed
Solution:	In standard include-file window3.i (derived from window2.i) the following should be added: if valid-handle(global-drill-handle) then ...
Warning :	VIEW-AS TEXT fields cannot be the only fields used in UPDATE statement (4017)
Problem :	All of the fields participating in a UPDATE statement are of type TEXT. Since TEXT widget are read-only and cannot accept any keyboard focus, the UPDATE statement can continue executing. This error occurs only when the input stream is directed to the terminal
Solution :	The converter has added 'report' attributes to an update-frame. The frame definition is often located in a separated include file. Changing the convert mode to maintenance could solve it. Note: If the include file is also used by report programs the convert mode change will not solve anything. It must be copied.
Warning :	** Incompatible data types in expression or assignment. (223)
Problem :	The converter activates the following source: if valid-handle(global-drill-handle) then global-drill-handle:screen-value = frame-value.
Solution :	Modify source to: if valid-handle(global-drill-handle) then global-drill-handle:screen-value = string(frame-value, global-drill-handle:format).

The positioning of frames and fields within frames could differ from the character-based version. Usually the vertical positioning of a frame will be handled by the sequence of viewing frames. Co-ordinates of a positioning are always integers and they can easily be checked.

In a GUI-environment the positioning of frames is more precise because frame-co-ordinates are decimal. A shining example of this is the use of address-info-frames, mfadform.i in program 10.3 cochmt.p.

This will work properly in a CUI environment:

```
{mfadform.i "sold_to" 1 " Verkoop "}
{mfadform.i "ship_to" 41 row 5 " Factuuradres "}
```

But fails under GUI circumstances;

Contract Algemeen Onderhoud

Gebruik menu Bewerken Eij Opties Help

PT opvragen SO opvragen PO opvragen GB (36.20.2)

Contractnr: 960030 ant: 1001000 Fakt.ad: 1001000 Verz.ad: 1001000

Verkoop

Reliable Central Purchasing
Attention: Accounts Payable
300000020 Century Blvd
Los Angeles CA 89000
U.S.A

Kt.ad/Verz.ad

Reliable Central Purchasing
Attention: Accounts Payable
300000020 Century Blvd
Los Angeles CA 89000
U.S.A

ALGEMENE CONTRACT INFO

Omschr: eee	Kanaal:	Commissie
Ref.Kint:	Taal:	Agent: JJP
Hoofd:	Bron:	Comm: 0.00%
Type:	Comment: nee	Meer: nee

Order: 01/10/96	Bevestigd:	Betalingsvw: 2/10-30	Blok: Nee
Start: 01/10/96	Origineel:	Prijslijst:	Valuta: YEN
Eind: 01/10/96	Ltst Ret: 01/10/96	BTW J/N: Nee 0	

LET OP: BEGIN BESTAND.

F1-? F2-OK Esc-Eind CTL+X-Knippen CTL+C-Kopiëren CTL+V-Plakken

This is solved by deleting 'row 5'.

