### Table of Contents

Published Standards ................................................. 9  
DST-0001 Designing for Translatable Strings .................. 10  
DST-0002 Designing MFG-PRO APIs .............................. 34  
DST-0003 MFG-PRO Conversion Guidelines - eB2 and below .......... 35  
DST-0005 Designing with OID fields (eB3 and up) ............ 56  
DST-0006 Design Considerations for Domained MFG-PRO Versions (eB2.1 and up) .................................................. 61  
STD-0004 SETUSERID use for Oracle ............................. 65  
STD-0006 Allowed number of fields per table .................. 66  
STD-0007 UNKNOWN values used in Unique Keys in Indexes Must Be Unique ......................................................... 67  
STD-0008 Indexes and Performance ............................... 70  
STD-0011 Maintaining Sequences ................................. 74  
STD-0012 Always Trim Substrings ................................ 78  
STD-0013 Table Name and Description ........................... 79  
STD-0014 Field Names ........................................... 82  
STD-0017 Using Shared frames, streams etc ..................... 86  
STD-0018 Use getFrameTitle() function to assign Frame Titles (eB and up) ......................................................... 89  
STD-0019 Report Header Problem ................................ 91  
STD-0020 Using Language Detail ................................ 93  
STD-0022 Database character set ................................ 96  
STD-0023 INPUT function on a variable referenced in multiple frames ................................................................. 99  
STD-0024 Expanding text for longer translation ................ 101  
STD-0025 Avoid string concatenation in Oracle ................ 102  
STD-0026 Don’t build strings to be displayed by concatenating .... 104  
STD-0028 Control File Labels .................................... 105  
STD-0029 Field Labels - Do not include punctuation marks .... 106  
STD-0030 Label Names ........................................... 107  
STD-0031 Report - Page Limits and Printing Interruption ...... 108  
STD-0032 Progress Schema Triggers .............................. 110  
STD-0034 Never set the mfguser variable ....................... 116  
STD-0036 Handling of Directory Names and Paths .............. 118  
STD-0037 Abbreviations for Field Labels ....................... 119  
STD-0039 Use of 'pause 0 before-hide' statement - pause 0 is preferred ................................................................. 121  
STD-0040 Ensure that a user can respond to 'Press space bar to continue' with either the spacebar or end-key (F4-esc) .... 122  
STD-0041 Use of user fields in MFG-PRO is NOT allowed ....... 124  
STD-0042 Obsoleting Fields .................................... 128  
STD-0043 Field Formats Allowed ................................ 130  
STD-0044 Definition And Acceptable Values For Logical Variables In Progress .......................................................... 132  
STD-0045 Translatable hard-coded strings should appear only in a
Possible values must be explicitly stated ............................... 232
STD-0149 Never abbreviate field or file names .......................... 234
STD-0150 Local Variable Naming Conventions ............................ 235
STD-0151 Frames in Maintenance programs - WIDTH .................... 237
STD-0153 Printing from a maintenance program .......................... 238
STD-0154 Prompts in batchable programs ................................ 240
STD-0155 Pop-Up Frames in Maintenance programs - add SPACE(2) ... 242
STD-0156 Frames in Maintenance programs - Positioning widgets .... 244
STD-0157 Frames in Maintenance programs - explicit SKIP ............ 246
STD-0158 Don't use VALIDATE. ........................................... 248
STD-0159 Frames in Maintenance programs - Positioning frames vertically .................................................. 250
STD-0160 Frames in Maintenance programs - CHOOSE ................. 252
STD-0161 Frames in Maintenance programs - FORM statements ...... 253
STD-0162 What the GUI converter does to Full GUI reports ........... 255
STD-0164 Read unique records - 'FIND' .................................. 257
STD-0165 Use OPEN QUERY-GET NEXT instead of FIND FIRST-FIND NEXT ....................................................... 259
STD-0167 Use TEMP-TABLES instead of xx_wkfl ....................... 261
STD-0174 Appropriate use of 'Define ... like' syntax. ................. 263
STD-0175 Put explicit SKIP at the last line when using PUT statement ......................................................... 265
STD-0178 Arrays must not be used if you need unique help or scrolling windows for each array element. ......................... 268
STD-0179 Do not use the 'Abbreviate Index' option ...................... 270
STD-0180 Every table must have a unique index & a primary index (pre-eB3) .............................................................. 271
STD-0182 Dates - Display format ........................................... 273
STD-0183 Sequence Naming ............................................... 275
STD-0187 Always specify no-lock or exclusive-lock .................... 277
STD-0188 Record Selection Proper Use of Indexes ...................... 279
STD-0189 Date field validation - Required dates ....................... 281
STD-0190 Dates - Adding to or subtracting from ....................... 282
STD-0191 Date range validation - Ending date must never precede the starting date ................................................... 285
STD-0192 Date range validation - Sequential ranges should not overlap ................................................................. 287
STD-0193 Validating GL Account Information (pre-eB) ............... 289
STD-0195 Source file formatting - Location of mfdtitle.i or mfdeclre.i . 291
STD-0196 Source file formatting - Header (pre-eB2) .................... 294
STD-0197 Use of comments formatting inside brackets of an include statement ......................................................... 300
STD-0198 Source file formatting - Patch markers ....................... 302
STD-0199 Always use a WHERE clause in queries against a database ................................................................. 303
STD-0200 Avoid 'NOT' operator in WHERE clause ..................... 305
STD-0261 Localization Tags Avoid placing tags in display statements 407
STD-0262 Localization Tags Standard for placing tags in find-for each-where statements .......................................................... 409
STD-0263 How to use the 'pxrun.i' construct .......................................................... 412
STD-0264 How to use the 'pxfunct.i' construct .......................................................... 418
STD-0265 Use 'pxrun.i' in XUI code to execute internal procedures .......................................................... 422
STD-0266 Use 'pxfunct.i' to call functions in XUI code .......................................................... 424
STD-0267 Use 'pxmaster.i' in all XUI Space Controllers .......................................................... 425
STD-0268 Use 'pxphdef.i' and 'pxgetph.i' to improve XUI performance .......................................................... 427
STD-0273 Proper use of Progress 'valid-handle' function with procedures .......................................................... 431
STD-0274 Do not use ' U' on literal strings .......................................................... 434
STD-0275 Use getTermLabel() function variants to access literal strings in the code (eB and up) .......................................................... 435
STD-0276 Source file formatting - no commented out code .......................................................... 438
STD-0277 Always use pxmsg.i when calling messages (eB and up) .......................................................... 439
STD-0279 Always define buffers for database table access within internal procedures .......................................................... 442
STD-0280 Do not use 'in this-procedure' on run statements .......................................................... 447
STD-0286 Index Names .......................................................... 448
STD-0288 Use lookups created via 'Browse Maintenance' - do not use scrolling window lookups .......................................................... 451
STD-0290 Do not use the Progress Browse widget in maintenance program frames .......................................................... 452
STD-0291 Do not separate labels from fields .......................................................... 454
STD-0292 Use gpbranch.p to handle branched lookups .......................................................... 456
STD-0293 Do not enable fields in more than one frame at a time .......................................................... 458
STD-0294 Modifying screens within an editing block or trigger .......................................................... 460
STD-0295 Storage Areas on tables and indexes .......................................................... 462
STD-0296 low_date and hi_date global variables - do not store in database .......................................................... 467
STD-0297 Reuse attributes of common schema fields .......................................................... 469
STD-0298 Use gpbrparm.i to pass parameters to lookups .......................................................... 473
STD-0300 Use gpwait.i to pause for display only frames .......................................................... 475
STD-0301 Oracle MAX-WIDTH (SQL-WIDTH) must be specified on new database fields .......................................................... 477
STD-0302 Define related fields with similar schema properties .......................................................... 480
STD-0303 Variables used to prompt for options .......................................................... 483
STD-0304 All CM-controlled files should have copyright & versioning .......................................................... 485
STD-0305 Extending frozen schema tables to provide extra QAD fields (eB2 and up) .......................................................... 487
STD-0306 Searching for field, table or index usage in MFG-PRO .......................................................... 492
STD-0307 Providing for Extra QAD Fields .......................................................... 493
STD-0308 Proper Use of qad_wkfl (eB2 and up) .......................................................... 495
STD-0310 Progress source code file names in MFG-PRO .......................................................... 500
STD-0311 Progress selection-list widget and the Desktop ................................ 502
STD-0312 Source file formatting - Header on files in RCS (eB2 and up) .................................. 505
STD-0313 Double-Byte Languages Issues with FORMAT, STRING & INDEX .................................. 512
STD-0314 Double-Byte Languages Issues with LENGTH, SUBSTRING & OVERLAY ................................. 514
STD-0316 Source file formatting - Header on files in Continuus CM (eB2 and up) ................................. 516
STD-0320 Domain Should Be Considered In All Queries (eB2.1 and up) ........................................... 519
STD-0322 Temp-table names and their field names should be 50 characters or less ............................. 523
STD-0323 Switching domains in procedure code ................................................................. 524
STD-0325 Use OID fields as foreign keys to relate qaddb tables (eB3 and up) ....................................... 527
STD-0326 Every qaddb table must have a create trigger (eB3 and up) ............................................ 532
STD-0327 Every table must have a unique index & a primary index (eB3 and up) ............................... 534
STD-0328 All field names must be qualified in LIKE clauses in mfdeclre.i ......................................... 538
STD-0333 Database fields with Extents are not supported ......................................................... 539
STD-0336 Unicode & Code Page-independent Programming Issues with LENGTH, SUBSTRING & OVERLAY ................................................................. 541
STD-0337 Issue with TEXT() option in SET, UPDATE and ENABLE statements ...................................... 545
STD-0340 Proper assignment of alias for Progress and Oracle databases ........................................... 547
STD-0341 Assigning correct value to 'hi_char' when code page is UTF-8 or collation is ICU .................... 549
STD-0343 Utilities - Suggested program structure ................................................................. 551
STD-0347 Source file formatting - Header of files in Subversion (SVN) eB3 and above ........................... 556
STD-0350 Maximum Record (Row) Size Limit ................................................................. 559
STD-0351 Supported Progress Data Types ..................................................................................... 562
Published Standards

The following standards are approved for use.
DST-0001 Designing for Translatable Strings

Summary

This document contains essential guidelines for the design of code that includes text to be translated by QAD prior to software release.

<table>
<thead>
<tr>
<th>ID</th>
<th>DST-0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Devmstr, Translatable Strings</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities for MFG/PRO versions eB2 and up</td>
</tr>
</tbody>
</table>

The source content for this standard was last updated on August 19, 2002.

General Principles and Guidelines

- All displayable text in a program must come from msg_mstr, lngd_det, or lbl_mstr. Messages come from msg_mstr. Strings that require mnemonics come from lngd_det. Labels come from lbl_mstr. Note: Menu titles come from mnt_det.

- Beginning with the eB2 Release, no new translatable string preprocessor variables should be introduced into the code. Any new local variables that are displayed in a frame must have the LABEL qualifier with the US English label specified in the DEFINE VARIABLE statement. Similarly, if a schema field is included in a FORM statement but the desired display label is different then the default schema label then the LABEL qualifier must be specified with the desired US English label. Label terms and label detail records are required for all variables displayed. For example:

```c
/* Local Variables */
define variable option_list as integer no-undo format "g"
   label "Options" initial 1.
form
   rst_desc colon 35
   rst_srvty_list colon 35
   skip(1)
   rst_notify1 colon 35 view-as fill-in size 35 by 1
      label "Users Notified At Submit"
   skip(1)
   rst_notify1_mail colon 35 view-as fill-in size 35 by 1
      label "Users E-mailed At Submit"
   skip(1)
   rst_notify2 colon 35 view-as fill-in size 35 by 1
      label "Users Notified At Approval"
   skip(1)
   rst_notify2_mail colon 35 view-as fill-in size 35 by 1
      label "Users E-mailed At Approval"
with frame b side-labels attr-space width 80.

/* SET EXTERNAL LABELS */
setFrameLabels(frame b:handle).
```

- All labeled frames (that is, side-label frames and down frames which contain at least one field with a label) must have a setFrameLabels() call.

- There can be no literal strings (for example, “Customer Address” to 54) in a frame’s definition. Otherwise, setFrameLabels() will not work. These literal strings should be defined as no-label variables and initialized.
to the value of the original literal string using the getTermLabel function. Subsequent DISPLAY statements
for the frame need to explicitly reference the variable.

- All fields that display a label within a frame that uses setFrameLabels() must be supported by Label Detail
  records.

- Label master and detail records are release specific (starting with eB). When making changes to the
  legacy code in prior releases that involve changes to the labels displayed in the UI, updates to the label
  master and detail devmstr records are required for each release effected.

- 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. This is not
  required for spare QAD and user fields which are not referenced in the code.

- All explicit use of label-retrieving functions (such as getTermLabel() and getFrameTitle()) must be
  supported by Label Master records.

- Don’t build sentences from separate text strings. Create new messages in msg_mstr for displaying
  informational messages as well as 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 the label functions within a program

gplabel.i contains all the definitions for the label-retrieving functions and must be included in each .p file that uses
a label function.

Any program that includes mfdtitle.i does not also need to include gplabel.i since gplabel.i is included in mfdtitle.i.

For any program that does not include mfdtitle.i, gplabel.i should be included as near as possible to the top of the
file (after the revision tags and, if available, after the mfdeclre.i call).

If mfdeclre.i is not included in the program, a definition for the shared variable global_user_lang_dir is required.

Include files should not include gplabel.i.

### Label Master and Label Detail Records

#### Label Master (lbl_mstr)

The Label Master table (lbl_mstr) stores all labels that appear on MFG/PRO’s screens and reports. A language
code and term identify each record in the table:

The **language code** (lbl_lang) indicates the language the record’s labels appear in (such as German or French).

The term (lbl_term) corresponds, roughly, to the meaning of a label. A term value has the convention that it is
composed **entirely of upper-case letters and underscores**. For example, SALES_ORDER_NUMBER. The
length of a term must not exceed 35 bytes.

The fields lbl_long, lbl_medium, lbl_short, and lbl_stacked contain the longest label (limited to 50 bytes), the
medium length label (limited to 30 bytes), the shortest label (limited to 20 bytes), and the stacked label (limited to
50 bytes with a bang (!) inserted between the upper and lower labels), respectively, for the term. The best approach
for creating new label master records is to use the ‘whole’ label as the long label even if you don’t currently have
room for it in the display; someone might be able to use the whole thing in another place in the UI and the
translators will be better able to accurately translate. The medium and short labels contain shortened or
abbreviated versions of the long label which can be utilize when the screen real-estate is limited.

The following is a list of rules for the long, medium, short, and stacked labels. Labels must:

- be in English
- be in mixed case (unless it consists only of an acronym)
- have each of its principal words begin with a capital letter
- begin and end with alphanumeric characters
- not contain two or more consecutive white spaces
- contain only alphanumeric characters and white spaces (but no punctuation marks)
- be at least 2 characters long
- contain only standard abbreviations and spellings
- not be used to combine with other labels to form longer strings
• not function as a message (messages are typically complete sentences, labels are typically short, descriptive text phrases)
• must be unambiguously associated with a Label Master term

Example:

<table>
<thead>
<tr>
<th>Term</th>
<th>“SALES_ORDER”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long Label</td>
<td>“Sales Order”</td>
</tr>
<tr>
<td>Medium Label</td>
<td>“Order”</td>
</tr>
<tr>
<td>Short Label</td>
<td>“SO”</td>
</tr>
<tr>
<td>Stacked Label</td>
<td>“Sales!Order”</td>
</tr>
</tbody>
</table>

Label Detail (lbld_det)

The 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. In addition, the Label Detail table will allow a term to be associated with a variable or field for a particular menu-level program.

The executable filename (lbld_execname) specifies the menu level program for the display variable. All labeled frame fields must be defined in lbld_det.

Guidelines for creating new lbl_mstr and lbld_det records

• Do not change the Long, Medium, Short or Stacked Label on an existing label without comprehensively researching what fields are using it and verifying that the change will not adversely effect any screens where that label is currently displayed.

• Determine if there is an existing term that can be used instead of creating a new term. If there is sufficient screen real estate available to support a more descriptive label and that label already exists in the label data then it should be used rather than create new label data records.

• All new label data must be submitted through devmstr and should be on the same ECO as the code that is introducing the new label. Label data is MFG/PRO version specific.

Devmstr Interface for lbl_mstr

The lbl_mstr table can be maintained through Devmstr using 36.7.9.1 The user interface is shown below:
Devmstr Interface for lbld_det

The lbld_det table can be maintained through Devmstr using 36.7.9.6. The user interface is shown below:
The label-retrieving functions

A label can display in a number of ways. For example, "Customer Address" can display as:

```
"Customer Address"
" Customer Address 
"    Customer Address
" Customer Address"
"    Customer Address:
```

or

```
---- Customer Address ----
```

These examples represent the most common situations. The following functions have been developed to accommodate each of the above types of strings. It is important to become familiar with the following 6 functions.

**getTermLabel()** returns a label for a specified term. It has two input parameters: p-term (the term) and p-length (an integer which represents the horizontal space, in bytes, which is available on the screen or report). The function will return a character string, to set the label attribute for a variable or field. The function determines which label to return (long, medium, short) based on the maximum length specified. If the length of the lbl_long label associated with the term exceeds the value of p-length then it checks the length of the lbl_medium label. If lbl_medium is also too long then it checks lbl_short. If all of the labels defined in the lbl_mstr record corresponding to the specified term have a length greater than the value of p-length or if lbl_long is too long but no lbl_medium or lbl_short labels exist then the function will truncate the label stored in lbl_long.

**Important:** You cannot use a getTermLabel or another function in the where clause of a for each, find or whatsoever. It will compile, however it generates a run-time error.

**Example:**

The following...
getTermLabel("CUSTOMER_ADDRESS", 30)

...returns:

"Customer Address".

Note that 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 case 16 bytes) for the specified term that does not exceed the length parameter (in this case 30 bytes).

Example:

Given the following lbl_mstr record...

| Term: | "SALES_ORDER" |
| Long Label: | "Sales Order" |
| Medium Label: | "Order" |
| Short Label: | "SO" |

The following...

getTermLabel("SALES_ORDER", 10)

...returns:

"Order".

and the following...

bel("SALES_ORDER", 4)

...returns:

"SO".

g ETFram et i tle() 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().

Example:

The following...

gETFram eTitle("CUSTOMER_ADDRESS", 30)

...returns:

" Customer Address ".

As with getTermLabel(), the string's overall length is restricted by the length parameter (here 30) but is not wholly determined by it.

g ETFram et Lable C entered() returns a label for a specified term, centered within the specified length. Same input parameters as getTermLabel().

Example:

The following...

gETFram eTitle("CUSTOMER_ADDRESS", 30)

...returns:

" Customer Address ".

Note that the string's length is determined entirely by the length parameter. The longest label available that does not exceed the specified length is used, and the label is centered in a string that is as long as the specified length.

gETFram et Lable F illC entered() 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.
Example:
The following...

dynamic-function ('getTermLabelFillCentered' in h-label,
           Input "CUSTOMER_ADDRESS",
           Input 30,
           Input "**")

...returns:

"****** Customer Address ******"

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.

getTermLabelRt() returns a right justified label for a specified label term. Same input parameters as getTermLabel().

Example:
The following...

getTermLabelRt("CUSTOMER_ADDRESS", 20)

...returns:

"    Customer Address".

This function behaves like getTermLabelCenter() with the difference that it right justifies instead of centering the label.

getTermLabelRtColon() returns a right justified label with an appended colon for a specified term. Same input parameters as getTermLabel().

Example:
The following...

getTermLabelRtColon("CUSTOMER_ADDRESS", 20)

....returns:

"   Customer Address:".

This function works the same as getTermLabelRt() but it embeds a colon in the string. This function is useful because label values in lbl_mstr do not contain colons. The colon takes up one of the bytes specified in the length parameter. In other words, the above returned string is 20 bytes long, colon included.

setFrameLabels()
In most cases it is unnecessary to explicitly invoke the label-retrieving functions listed in the last section. 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 2 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.

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. 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). Note that "buffer" frames, which don’t, typically, display, should also NOT have a setFrameLabels() call. A buffer frame is a frame that contains a table or buffer variable only.
For each frame that requires setFrameLabels(), determine where to place the 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.

setFrameLabels() should appear after the frame definition of a FORM statement or, in cases where the frame is defined outside of a FORM statement, after the frame definition but before its first DISPLAY. In general,

- each displayed frame must be named,
- each named frame needs to have only one call to setFrameLabels(), and...
- each setFrameLabels() call must appear within the scope of the named frame.

**Example 1:**
If the frame is defined by a FORM statement, insert setFrameLabels() directly after the last line of the “with frame” statement:

```plaintext
form
  so_nbr colon 15
  with frame a side-labels width 80.
/* SET EXTERNAL LABELS */
setFrameLabels(frame a:handle).
```

**Example 2:**
If the frame is defined within a FOR EACH loop, insert setFrameLabels() within the loop. For example:

```plaintext
for each x with frame b:
  statement.
  /* SET EXTERNAL LABELS */
  setFrameLabels(frame b:handle).
  display x.
end.
```

(Note that 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.)

**Example 3:**
If the frame is defined within the branch of a conditional, insert setFrameLabels() within the conditional branch. As above, this only applies for cases where the frame is actually defined in the branch and not before it. Whenever possible, avoid making multiple insertions of setFrameLabels in each sub-loop or conditional branch. If it is necessary, then insure that all branches of the logic that involve DISPLAY statements are being properly handled. For example:

```plaintext
if ... then 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.
```

In any cases where the frame is defined in the conditional display, the above constructs should be used. For frames not defined previous to the display, the following constructs should be avoided:

```plaintext
if ... then display x.
```

and

```plaintext
else ... display x.
```

These constructs allow no place for setFrameLabels() and should be rewritten to match the constructs above, or, a better solution, define the frame in a FORM statement at the beginning of the file or program.

**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. For example:

```plaintext
display x with frame b.
```

should be rewritten as:

```plaintext
do with frame b:
  /* SET EXTERNAL LABELS */
  setFrameLabels(frame b:handle).
  display x.
end.
```

**How To: Frames which contain Button widgets**

Buttons should be defined as "side label" type widgets. Progress allows you to use them in a down frame, but, because the algorithms used in setFrameLabels, they should only be defined within side-label frames.

```plaintext
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).
```

**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:

```plaintext
define variable StaCmf like mfc_logical format "Accepted/Rejected"
  no-undo.
```

The following Label master record must be defined:

<table>
<thead>
<tr>
<th>Term:</th>
<th>&quot;ACCEPTED/REJECTED&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long Label:</td>
<td>&quot;Accepted/Rejected&quot;</td>
</tr>
<tr>
<td>Medium Label:</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>Short Label:</td>
<td>&quot;A/R&quot;</td>
</tr>
<tr>
<td>Description:</td>
<td>Logical Format</td>
</tr>
</tbody>
</table>

**How To: Handling character variables which contain literal text**

1. 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**

   ```plaintext
define variable del as character format "x(15)".
   del = getTermLabel("DELETED",15). /* ASSIGN INITIAL VALUE */
```

2. 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.

   **Example**
define new shared variable sortextoption as character extent 2 format "x(58)".

/*************** SET INITIAL VALUES *************** */
assign
    sortextoption[1] = "1 - " + getTermLabel("BY_SHIP-FROM",12) + ", " +
                        getTermLabel("CUSTOMER",8) + ", " +
                        getTermLabel("SHIP-TO",7) + ", " +
                        getTermLabel("ORDER",5) + ", " +
                        getTermLabel("DOCK",4) + ", " +
                        getTermLabel("ITEM",4).
    sortextoption[2] = "2 - " + getTermLabel("BY_SHIP-FROM",12) + ", " +
                        getTermLabel("ITEM",4) + ", " +
                        getTermLabel("CUSTOMER",8) + ", " +
                        getTermLabel("SHIP-TO",7) + ", " +
                        getTermLabel("ORDER",5) + ", " +
                        getTermLabel("DOCK",4).

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}
    shipto_from colon 15
    shipto_to colon 45 label {t001.i}
    part_from colon 15
    part_to colon 45 label {t001.i}
    po_from colon 15 format "x(22)"
    po_to colon 45 label {t001.i} format "x(22)"
    skip(1)
    schtype colon 15
    current_ind colon 15
    sortoption colon 15
    skip
    sortextoption[1] at 17 no-label
    sortextoption[2] at 17 no-label
    skip(1)
with frame a side-labels width 80 attr-space.

/* SET EXTERNAL LABELS */
setFrameLabels(frame a:handle).

3. If the FORM or frame contains a string that has been set by the ENTRY function, use getTermLabel() for each such string.

Example 1

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
    ln_site @ seq_site
    ln_line @ seq_line
    last_date @ seq_due_date
    entry(weekday(last_date),disp-wdays) @ daylabel.

Example 2
define variable xfer-option-input as character label "Cost Transfer Option" format "x(36)" no-undo.
define variable xfer-option-list as character no-undo.
define variable xfer-option-integer as integer initial 1 no-undo.

xfer-option-list = "1-" + getTermLabel("COMBINE_OLD_AND_NEW_ELEMENT_COSTS", 35) + ","
                   + "2-" + getTermLabel("OLD_ELEMENT_COST_ONLY", 32) + ","
                   + "3-" + getTermLabel("NEW_ELEMENT_COST_ONLY", 33).

xfer-option-input = entry(xfer-option-integer, xfer-option-list).

form
csset colon 25
description no-label
skip(1)
   element-from colon 25 descript-from no-label
   element-to colon 25 descript-to no-label
skip(1)
   part colon 25 part1 label {t001.i} colon 50
   site colon 25 sitel label {t001.i} colon 50
skip(1)
   xfer-option-input colon 25
   update_yn colon 25
with frame a side-labels width 80 attr-space.

/* SET EXTERNAL LABELS */
setFrameLabels(frame a:handle).

4. Be aware that you cannot use a user defined function (like getTermLabel) within an ASSIGN statement after the assignment of the key fields. For example, the following ASSIGN statement...

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).

...will generate the error message:

Attempt to reference user-defined function 'getTermLabel' in an ASSIGN type statement after an assignment to key field 'rnd_mstr.rnd_rnd_mthd'. (7955)

To avoid this problem, either:

(A) 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:

(B) Do all key field assignments in a separate ASSIGN statement:
assign
  rnd_unit = 1
  rnd_threshd = 0.5
  rnd_dec_pt = "."
  rnd_desc = getTermLabel("ROUND_TO_ONES",16).
assign
  rnd_rnd_mthd = "0"

How To: Handling literal strings in frames

1. 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, first define the literal string as a character variable. Next set the initial value of the variable using getTermLabel(). Finally, reference the variable name in the FORM or DISPLAY statements, as appropriate. The NO-LABEL qualifier should be specified in either the DEFINE statement or subsequent FORM statement.

Example

```qade
define variable c-self-bill-line as character format "x(22)" no-undo.
define variable c-shipment as character format "x(19)" no-undo.
define variable c-discrepancy as character format "x(15)" no-undo.

c-self-bill-line = getTermLabel("SELF-BILL_LINE",14) + fill("-", 8).
c-shipment = getTermLabel("SHIPMENT",8) + fill("-", 11).
c-discrepancy = getTermLabel("DISCREPANCY",11) + fill("-", 4).
```

```qade
form
  sbi_bill colon 16
  c-self-bill-line no-label at 46
  c-shipment no-label at 87
  c-discrepancy no-label at 111
  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
  sbi_bill
  c-self-bill-line
  c-shipment
  c-discrepancy
  sbi_xmission
  sbid_qty
with frame f_detail.
```

2. An alternate approach is to make use of the screen-value attribute to set the screen value of the character variable as part of the frame definition. However, this technique is not recommended when the same frame is to be displayed multiple times during program execution. Using the screen-value attribute will only display these labels the first time the frame is brought into view. If there are any CLEAR FRAME statements then you lose the labels assigned using the screen-value attribute if the frame is then redisplayed. One of the benefits of this technique is that the DISPLAY statements for the frame do not need to include all the character variables used for the literal text. For example, in the following example a VIEW FRAME m1 will display the frame and all the literal strings. Whereas, using the method shown above a DISPLAY statement would be required to display the values of each of the character variables.

Example

```qade
define variable c-self-bill-line as character format "x(22)" no-undo.
define variable c-shipment as character format "x(19)" no-undo.
define variable c-discrepancy as character format "x(15)" no-undo.
c-self-bill-line = getTermLabel("SELF-BILL_LINE",14) + fill("-", 8).
c-shipment = getTermLabel("SHIPMENT",8) + fill("-", 11).
c-discrepancy = getTermLabel("DISCREPANCY",11) + fill("-", 4).
```
form
" 1 -" lbl-del-shipper format "x(50)" skip
" 2 -" lbl-del-item format "x(50)" skip
" 3 -" lbl-del-cont format "x(50)" skip
" 4 -" lbl-rem-cont format "x(50)" skip
" 5 -" lbl-rem-cont-plus format "x(50)" skip
with frame m1 width 80

3. For HEADER frames the literal strings can be included in the frame definition using the getTermLabel() function. However, there is a performance penalty for doing so. If a function is used in a FORM statement, the function will be called each time when the data is displayed which is a costly and unnecessary operation.

Note: There is a known Progress bug involving page-bottom header frames. Progress produces segment fault error at runtime when any user defined function is used in header frames with the "page-bottom" attribute. According to Progress, this bug should be fixed in Progress version 9.1c.

Example
form header
mc-curr-label et_report_curr skip
mc-exch-label mc-exch-line1 skip
mc-exch-line2 at 23 skip(1)
(getTermLabel("VOUCHER",12)) format "x(12)"
(getTermLabel("INVOICE_DATE",8)) format "x(8)"
(getTermLabel("EFFECTIVE_DATE",8)) format "x(8)"
(getTermLabel("CREDIT_TERMS",8)) format "x(8)"
age_range[1] space (4)
age_range[2] space (1)
age_range[3] space (1)
age_range[4] skip(0)
(getTermLabel("INVOICE",12)) format "x(12)"
(getTermLabel("DUE_DATE",8)) format "x(8)"
(getTermLabel("EXCHANGE_RATE",8)) format "x(8)"
(getTermLabel("CURRENCY",8)) format "x(8)"
(getTermLabelCentered("DAYS_OLD",17)) format "x(17)"
(getTermLabelCentered("DAYS_OLD",17)) format "x(17)"
(getTermLabelCentered("DAYS_OLD",17)) format "x(17)"
(getTermLabelCentered("DAYS_OLD",17)) format "x(17)"
(getTermLabelRt("TOTAL_AMOUNT",17)) format "x(17)"
skip

How To: Passing literal strings as arguments to Include Files

There are several commonly used include files that have been modified as part of the Labels Project. Previously, these include files required that a literal string be passed using a preprocessor variable. Now the literal strings should be passed using the appropriate label master terms (all caps, underscores between words). Below is a table listing some of the effected include files. Any new include files which require literal strings for arguments should be implemented in a similar fashion.

<table>
<thead>
<tr>
<th>Include File</th>
<th>Literal String Argument</th>
<th>How literal is used in Include file</th>
<th>Examples</th>
</tr>
</thead>
</table>

22
### How To: Display Frame Titles

1. Use `getFrameTitle()` to display the frame title.
2. Specify the maximum length possible for the frame title.

#### Dynamic Frame Titles

Dynamic frame titles that 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 `getFrameTitle` to retrieve the desired frame title.

#### Example 1
define variable frametitle-so as character format "x(20)"
define variable frametitle-rma-iss as character format "x(20)"
define variable frametitle-rma-rec as character format "x(20)"

/* SET INITIAL VALUES------- */assign
frametitle-so = getFrameTitle("SALES_ORDER_LINE",20)
frametitle-rma-iss = getFrameTitle("RMA_ISSUE_LINE",20)
frametitle-rma-rec = getFrameTitle("RMA_RECEIPT_LINE",20).

if this-is-rma then
  if rma-issue-line then
    frametitle = frametitle-rma-iss.
  else
    frametitle = frametitle-rma-rec.
else
  frametitle = frametitle-so.

Example 2
define input parameter usage-for-op like wod_op.
form
  wod-op column-label "PAO Line"
  pjs-line label "Project Line"
  wod_part
  wod_ca_int_type label "Wk Code"
  wod_qty_req label "Qty to Issue"
  return-item label "Rtn"
  detail label "Det"
with frame f title color normal
if usage-for-op = ? then (getFrameTitle("ITEM_USAGE",16))
else (getFrameTitle("ITEM_USAGE_FOR_OPERATION",32)) +
  string(usage-for-op)
width 80
down attr-space.

/* SET EXTERNAL LABELS */
setFrameLabels(frame f:handle).

How To: Fields that have multiple labels within a single menu item

Try to avoid this practice when writing new code. Use different variable names when you need to display the same field in different frames with different labels.

Example 1
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. Follow this example:
if eximmode = &IMPORT then run qqimp 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
  qqError format "zz,zzz,zz9" colon 70 label &qqierp_p_19 skip
  qqTotal format "zz,zzz,zz9" colon 70 label &qqierp_p_20 skip
with frame ImpStats side-labels width 150.

/* SET EXTERNAL LABELS */
setFrameLabels(frame ImpStats:handle).

END PROCEDURE. /* qqimp */

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
  qqError format "zzz,zz9" colon 70 label &qqierp_p_24 skip
  qqTotal format "zzz,zz9" colon 70 label &qqierp_p_25 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)
    qqError:label = getTermLabel("NUMBER_OF_DOCS_NOT_SENT_DUE_TO_ERR", 68)
    qqTotal:label = getTermLabel("NBR_OF_DOCS_NOT_SELECTED_TO_SEND", 68)
END PROCEDURE.

Example 2

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) .

Example 3

If there are fields in the frame that have their label attributes set after the frame definition, use getTermLabel(). For example:

so_nbr:label in frame a = getTermLabel("MY_LABEL",23).
Progress does not allow one to set the column-label attribute dynamically. This means that the approach of using the "label = setTermLabel" construct only works for side-labels or for column labels in which only one row is used (and no labels are stacked).

How To: Handle PUT statements

A PUT statement requires that a format be passed to it for the string. Otherwise the PUT statement only reserves 8 bytes (Progress’s default length for a character string) for the string’s display and will truncate longer string at 8 characters. For any string that exceeds 8 characters, it is necessary to pass the length to the PUT statement.

Example 1

```plaintext
put "---------------" to 107
"---------------" to 132
skip(0)
string(et_report_curr + " "+
getTermLabel("SALESPERSON_TOTAL", 60) + ":")
format "x(25)" to 70.
```

Example 2

If the length available for the string being PUT is not fixed, it is necessary to have a dynamic calculation of the string’s length. This allows a string to be longer or shorter depending on its translation. In these cases, use gplblfmt.i. This file calculates the format of the string based on the label retrieved and uses that for the format length for the PUT statement.

```plaintext
if multi-due then
  put invoice at 1
  {gplblfmt.i &FUNC=getTermLabel(""MULTIPLE"",10) } at 14.
```

getTermLabel within gplblfmt.i requires two pairs of double quotes.

Example 3

```plaintext
if pod_rev <> "" then
  put {gplblfmt.i &FUNC=getTermLabel(""REVISION"",16)
  &CONCAT="":"} at 67 pod_rev.
if pod_disc_pct <> 0 then
  put {gplblfmt.i &FUNC=getTermLabel(""DISCOUNT_PERCENT"",9)
  &CONCAT="":"} to 99 pod_disc_pct.
if desc2 <> "" then
  put desc2 at 18 format "x(49)".
if pod_site <> po_site then
  put {gplblfmt.i &FUNC=getTermLabel(""SITE"",8)
  &CONCAT="":"} at 18 pod_site.

find first poc_ctrl no-lock no-error.
if available poc_ctrl and poc_ers_proc = yes
then do:
  put getTermLabel("ERS_OPTION",10) + ": "+
  string(pod_ers_opt) format "x(14)" at 33
  getTermLabel("ERS_PRICE_LIST_OPTION",27) + ": "+
  string(pod_pr_lst_tp) format "x(25)" at 48
  skip.
end.
```

How To: Displays with ACCUM, SUM, AVG, or Similar Keywords

For any calculations such as ACCUM, SUM, and AVG you wish to display the resulting calculation with a
label:

1. Define variables to store the calculations being performed.

2. 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.

Example

```qbasic
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"
```

How To: Include paragraphs of Text

The Label Project has resulted in the discontinuation of the t###.i files. Though these files will continue to be used for Releases prior to eB, they should not be used for eB and above.

Exception: {t001.i} and {t002.i} will remain in the eB files (To and From)

Exception: {t042.i} and {t043.i} will remain in the eB files because the are included in mfdeclre.i

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 the Devmstr program dvpcmmt.p.

Gpcdget.i should be included prior to the form/display statement for any input frame in the source file of a utility program.

Example: utbrgnmnt.p

```qbasic
/* DISPLAY TEXT FRAME*/
{gpcdget.i "UT"}

/**************************
   /* 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.
   */
```

The “UT” parameter 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.
There may be cases where there are multiple lines of text which need to be translated together but are not used in a utility program. In these cases, the "TX" parameter should be passed to the include file. This will retrieve the text from cd_det, but it is the parent program’s responsibility to display the data retrieved.

Example: povedrp2.p displays the following text at the bottom of report pages:

```
/*N0R7*/ define variable lit-text as character format "x(76)" extent 2 no-undo.
/*N0R7*/ {gpcdget.i "TX"}
/*N0R7*/ assign
    lit-text[1] = util-cmnt[1]

[...]

display
/*N0R7*/ {t015.i} */
/*N0R7*/ lit-text[1] at 2 no-label
/*N0R7*/ skip
/*N0R7*/ lit-text[2] at 2 no-label
    with frame c column 20 width 132.
```

Use the devmstr program: 36.7.12 dvgpcmmt.p to enter the literal text for the source file. The input screen is shown below:

When this program (utbrgnmt.p) is run from the menu, the following is the display:
How To: Displays Banners (Text strings that contain spaces between each letter)

Oftentimes in MFG/PRO when a 'title' is displayed on a printed form or report, there are spaces between each letter. This is intended to help it stand out more. For example:

"S A L E S O R D E R"
"P A C K I N G L I S T"
"S E Q U E N C E P A C K L I S T"
"M A S T E R B I L L O F L A D I N G"

These are referred to as banner labels. Banner labels are stored in label master records that begin with the prefix "BANNER_". For example:

Term: BANNER_SALES_ORDER
Long Label: S A L E S O R D E R

⚠️ Capitalization of each letter in the long label is allowed for banner labels.

Example
The include file pobl03h.i defines the header FORM for a Blanket PO:
How To: Stack Labels in a Down Frame

The externalized label functionality cannot properly handle stacked labels where the frame definition specifies two or more rows of stacked variables with different labels for each row. In other words, the frame definition includes a variable in the first line (with single regular label) then a different variable in the second line at the same position as first one with a different label. The intended effect is to stack the labels and the data being displayed. The setFrameLabels function will display the labels for the first row properly but it can’t figure out the labels for the second or subsequent rows.

To properly display the labels for stacked data in a down frame, the FORM statement should have just one row in the header and make use of stacked labels. Then there should be two (or more for multiple rows) DISPLAY statements – one for the first row of data and the second for the second row of data.

Example
The labels for the following frame will not display properly. This down frame contains two rows with different labels in each row.

```assembly
form
header   skip (3)
billto[1] at 4
getTermLabelRt("BANNER_BLANKET_ORDER",35) to 80 format "x(35)"
billto[2] at 4
/* DISPLAYS "SIMULATION" TEXT, IF REPORT IS RUN IN SIMULATION MODE */
if not update_yn and po_print then
  getTermLabel("BANNER_SIMULATION",28)
else
  ""
  at 43 format "x(28)"
billto[3] at 4
getTermLabelRtColon("BLANKET_ORDER",14) to 56 format "x(14)"
po_nbr
...
```

The correct approach is to define the frame with only one row of variables and to use stacked labels for each column, as appropriate. Then the data can be displayed using multiple DISPLAY statements as illustrated below:
/* FRAME C2 : FRAME TO DISPLAY APPROVER INFORMATION */

form
rqa_apr at 1 column-label "User!User Name"
var_apr_level at 31 column-label "Appr Lvl!Rvw Lvl"
var_apr_level_amt at 41 column-label "Appr Amt!Rvw Amt"
rqa_apr_req at 63 column-label "Appr Req!Rvw Req"
rqa_sub_from at 73 column-label "From Sub!To Sub"
rqa_cc_from at 83 column-label "From CC!To CC"
rqa_start at 91 column-label "Start!End"
rqa_category at 101 column-label "Catg!Type"
rqa_entity at 111 column-label "Entity!Job"
rqa_site at 121 column-label "Site!PL"
with frame c2 down width 132.

and the corresponding display statements are:

display
rqa_apr
rqa_apr_level
var_apr_level_amt
rqa_apr_req
rqa_sub_from
rqa_cc_from
rqa_start
rqa_category
rqa_entity
rqa_site
with frame c2 down width 132.

down 1 with frame c2.

display
var_apr_name format "x(30)" @ rqa_apr
rqa_rvw_level @ rqa_apr_level
var_rvw_level_amt @ var_apr_level_amt
rqa_rvw_req @ rqa_apr_req
rqa_sub_to @ rqa_sub_from
rqa_cc_to @ rqa_cc_from
rqa_end @ rqa_start
trim(substring(var_type_desc,1,1)) format "x(1)"
@ rqa_category
rqa_job @ rqa_entity
rqa_prod_line @ rqa_site
with frame c2 down width 132.

down 1 with frame c2.

setFrameLabels() and its internal procedures

setFrameLabels() is used to set the label attributes on fields within a frame. It takes the frame’s handle as its input. It runs internal procedures based on the frame type (side-label, down frame, etc.).

setSideLabels() sets translated labels for side-label frames. Used by setFrameLabels(). This function accepts as input a frame handle. It walks the frame’s widget tree to identify any widgets that have side-labels. If a widget is determined to be a a side label, the field’s translated label is retrieved and set via getFieldLabel(). If a widget is determined to be a variable with a text display value, then its label is retrieved and set via getFieldLabel().

setColumnLabels() sets translated labels on frames using column labels. Used by setFrameLabels(). Its input parameter is p-frame (handle of frame to set labels on). This function accepts as input a frame handle. The function walks the frame’s foreground frame group to identify how many columns the frame has and what the coordinates are for each column (the function uses the field’s format in this calculation). Once the number of columns and their coordinates have been determined, the function walks the frame’s background frame group to identify literal widgets that are used to display column labels. The literal widgets are expanded to the full format size to maximize real estate for translated labels. The translated label is then retrieved and set for each widget on the frame.
getFieldLabel() returns a label for a specified field. Its input parameters are p-field (field to get label for) and p-length (number of bytes the string is limited to). If no label is found, a "^" is returned.

getStackedLabel() returns the stacked label for a specified field. This procedure is used by setColumnLabels(). Its input parameter is p-field (the field to get the stacked label for) and its output parameters are p-label1 (string returned as top label of a stacked label), p-label2 (string returned as middle label of triple-stacked label or bottom label of stacked labels), and p-label3 (string returned as bottom label of triple-stacked labels). If no label is found c-term will be returned as top and bottom label.

setStackedColumnLabels() identifies literal widgets used for stacked column labels and sets them to input values. Its Input parameters are p-frame (frame handle), p-widget (widget labels are being set for), p-column (column number widget is representing), p-label1 (string used as top label), p-label2 (string used as middle label for triple-stacked or bottom label of stacked labels) and p-label3 (string used as bottom label of triple stacked labels).

justifyLabelByDataType() right justifies label according to the data-type of the widget the label is being set for. Its input parameters are p-widget (widget that label is being set for) and p-realestate (character length available for label). Its input-output parameter is p-label (actual label). This function is used primarily for by setColumnLabels(), as side labels are always right justified.

centerLabel() centers label according to character length available for label and actual length of label. Its input parameter is p-realestate (character length available for label). Its input-output parameter is p-label (actual label).

rightJustifyLabel() right-justifies label according to character length available for label and actual length of label. Input parameter p-realestate (character length available for label). Input-Output parameter p-label (actual label).

getFormatLength() calculates format length in bytes. Input parameter is p-widget (handle to widget) and output parameter is p-length (integer length of format).

erAvailableSideLabelSize() calculates maximum available size (in bytes) for a label. Input parameter is h-lebelledwidget (handle to widget that is displaying string on the UI) and output parameter is p-available (integer length of available space for label). Used by setSideLabels().

setBrowseLabels() sets translated labels on browses. Input parameter is p-browse (handle of browse to set labels on).

setMenuLabels() sets translated labels on menu bars. Input parameter is p-menu (handle of menu to set labels on).

setRadioSetLabels() sets labels on radio-set widgets. Input parameter is p-widget (widget labels are being set for).

clearFrameRegistration() clears the "private-data" value form the frame. The frame registration prevents frames from being translated more than once within an application to improve performance for programs that are likely to display several iterations of a frame (in, for example, a report). The function's input parameter is p-frame (handle of frame to clear registration on).

checkFrameRegistration() checks the frame registration to see whether a frame has been translated for a particular program. Its input parameter is h-frame (frame handle). Its output parameter is p-frame-registered (logical set to true denoting frame translation has already been performed).

addFrameRegistration() Assign the "private-data" value for frame to "TRANSLATED", denotes that this frame has been translated. The functions input parameter is h-frame (handle of frame to add frame registration).

getTranslateFramesFlag() returns a logical value of true if "Translate Frame" setting is turned on, false if not. Translate Frames is a control file setting that determines whether setFrameLabels() sets label values or not.

Steps to create XX language environment (Character)

xxx ======> means your three letter login
xx ======> means foreign language

1> cd/users/xxx/work91 ======> xxx means your three letter login

2> ln -s /qad/mfgpro/91/stageobj/us us

3> ln -s us xx

4> rundb d91

5> in progress editor, run xxlblmstr.p from /users/azr/docs to populate xx language.

6> copy mnt_det.d from /qad/mfgpro/91/stage/mtg to temp directory. Change the language code “us” to “xx” and load in to database.

7> run mf.p and select option 36.4.1 Language Maintenance
create language like:
language ID: xx
Directory: xx

8> run mgurmt.p from MFG/Pro menu and change language for your three letter login (xxx) to xx.

9> logout from MFG/Pro and rellogin. You will see all labels except Menu in xx.
If you don't see any labels in xx, then run gpiblp1.p and set setting "Translate Frames" to yes.

See Also

N/A
DST-0002 Designing MFG-PRO APIs

Summary

This document contains essential guidelines for designing and implementing Application Program Interfaces (APIs) into MFG/PRO.

<table>
<thead>
<tr>
<th>ID</th>
<th>DST-0002</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>APIs, MFG/PRO Design</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities for MFG/PRO Version 9.0, eB, eB2</td>
</tr>
</tbody>
</table>

Description


MFG/PRO API Guide (QDN Download - QAD Internal Only)

Rationale

N/A

See Also

N/A
DST-0003 MFG-PRO Conversion Guidelines - eB2 and below

Summary

Guidelines for developers of conversion programs.

| ID     | DST-0003 |
|--------|--|---|
| Version| 0 |
| Language| Progress |
| Status | Published |
| Categories | Conversions |
| Applicability | MFG/PRO Conversions Programs written for eB2 and below to convert data from previous versions |

Description

This design standard provides guidance for developers of ‘SV’ (source version) and ‘TV’ (target version) conversion programs. SV programs are used in place of the standard dump programs to generate custom formatted ".d" file(s) that conforms to the schema of the corresponding table(s) in the target version. TV programs operate on the records in the target version database to move data between fields, possibly in different tables, so that it fits the data model of the new version of the database.

Both the SV and TV programs are run from the conversion shell. The attached document also describes how the conversion shell functions so that maintenance can be done on the shell itself.

Purpose

This design standard provides guidelines for conversion developers for implementing conversion programs for converting data from previous MFG/PRO releases to MFG/PRO eB and eB2. See DST-0004 for eB2.1 and later.

Scope/Applicability

This process is applicable for anyone in the QAD R&D organization who needs to develop conversion programs.

Overview

This design standard provides guidelines for maintenance and enhancement of the MFG/PRO Conversion Shell, MFG/CONV. This document assumes that all the documentation for products requiring data conversion have been read, and that the overall MFG/PRO conversion process is understood. For more information on MFG/CONV and MFG/PRO conversions, refer to Conversion in Section 3 of the MFG/PRO eB Installation Guide.

Note that the term “conversion” includes the Conversion Shell, the conversion code, and frequently the “action” of performing a database conversion.

The Conversion Shell, MFG/CONV provides a single location for the integration of all MFG/PRO related conversion code. It also provides a single user interface and launch point for the multiple conversion programs available to MFG/PRO customers. The conversion code, created by a project team for their specific product, enables an existing database to be migrated to a more recent schema. During the initial planning and design of a product, Systems Engineering and/or the Architects may be consulted on conversion issues.

Developers who create conversion code are required to supply proof of an error-free unit-test before integrating their code into the Conversion Shell environment. Once the conversion code is integrated, the project team developer tests the conversion within the Conversion Shell environment using the appropriate test plan (refer to the Conversions Test Plans for UNIX and NT). Any errors in data conversion should be corrected and the cycle repeated until no errors are generated. Individuals who have been identified for further conversion testing, and who, ideally, were not involved in the product’s conversion code development, can use this test plan to ensure that proper testing is thoroughly performed. Conversion test results will be analyzed by the developer and signed off. This process enables the conversion code to be confidently moved into a release.

The Goal of Conversions
The conversion process as facilitated by the shell, MFG/CONV, assists the end user in migrating an MFG/PRO database from one schema to another. This migration is typically due to an upgrade from an earlier to a more current version of MFG/PRO. The Conversion Shell and the procedures it launches must run in all MFG/PRO supported environments. Currently, the conversion programs are English only.

Types of Conversions

The following conversion types are supported

**Full dump and load conversion** Data is dumped from the source database and loaded into the target database. Data can either be reformatted by special programs during the dump procedure or be converted after it is loaded into the target database.

**Incremental conversion** The source version database’s schema is updated and the data is converted “in place”, the converted source database then becomes the target database. A small amount of data that must be manipulated to prevent uniqueness violations in new indexes is still dumped and reloaded. Incremental conversions are currently supported for conversions to eB and eB2 as a services offering.

MFG/PRO Versions Supported

The eB and eB2 MFG/PRO versions currently support conversions from the releases shown below.

<table>
<thead>
<tr>
<th>From...</th>
<th>To eB</th>
<th>To eB2</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.3</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>7.4</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>8.5</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>8.6</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>8.6E</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>9.0</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>eB</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Any version of MFG/PRO prior to 7.3 must be converted to 7.3 or 7.4, before using the Conversion Shell to migrate to eB and eB2.

Progress to Oracle Conversions

Oracle conversions are supported. A customer may convert from Progress to Oracle or from Oracle to Oracle. Converting from Oracle to Progress is not supported.

Conversion Programs from a Project Team

System Engineering does not design, code, or debug conversion procedures for an MFG/PRO project team or bolt-on product, but rather maintains the shell, MFG/CONV, and its development environment. The shell provides an integrated interface from which the project teams’ conversions are launched. The developers create and debug the procedures required to perform the data conversion for their product. To ensure that the Conversion Shell calls these procedures in the proper order, developers must adhere to standard rules and practices. System Engineering is not responsible for understanding or supporting the programs that are received from a developer, but is responsible for ensuring the developer’s conversion programs function within the shell following the guidelines stated in this document.

System Engineering may consider consolidating two or more conversion programs supplied by a developer if the programs touch the same tables and massage the same data. Also, as time permits, System Engineering may consider process changes to improve the overall performance of the conversion.

Target Version Program Characteristics

Target version programs may manipulate huge amounts of data. For this reason, they implement the update as a series of large transactions. When something goes wrong, only the last transaction group is rolled back. This means that all target version programs must be re-runnable and re-startable. That is, if the work of the program does not finish, it must be able to run again and do the remainder of the work without disturbing the work already done. If the program does complete its work, it should be able to be run again and not harm the work already done. This can be done either by allowing the program to run and not disturb the data already converted, or by refusing to
run by recognizing that the conversion has been completed. Ideally all target version programs should possess both properties. That is, they should not run when they recognize that they have already run and even if they do run they should do no harm.

**MFG/PRO Conversion Program Standards**

**MFG/PRO Includes**

Each conversion program must run stand-alone from MFG/PRO, and MFG/PRO include files (programname.i) should NOT be used. If an include must be used, copy/change the include file to work inline and rename the file with a unique name. The include file must be delivered to the System Engineering with the conversion program(s). Ensure that the include file delivered does NOT use any other standard MFG/PRO include file or reference any MFG/PRO standard global variable.

As an example, if you would like to use the MFG/PRO mfgtime.i file in your program.p file: Make a copy of the mfgtime.i file to a different name, say convtime.i. Where the mfgtime.i file is needed in your convprogram.p file, include the convtime.i instead. The convtime.i file must be delivered with your convprogram.p file.

**Global Variables**

Do not use or rely on mdeclre.i.

**Use of gprun.i**

Do not use gprun.i, use run. The conversion software will make sure the correct PROPATH is set.

**Standard Programs**

If a standard MFG/PRO program must be run as part of the conversion program, copy it and give it a unique name. Each conversion program must run independent of MFG/PRO, and MFG/PRO include files (programname.i) should NOT be used. Ensure that the copied MFG/PRO program file delivered does NOT use any other standard MFG/PRO include file or reference any MFG/PRO standard global variable.

**Target Version Programs**

**Input Parameters**

The following input parameters must be created.

- DEFINE INPUT PARAMETER main AS WIDGET-HANDLE . (If a conversion program runs another program, the parameter 'main' must be passed in the argument string.)
- DEFINE INPUT PARAMETER basever AS CHARACTER . This parameter is used to determine the version of MFG/PRO the conversion is converting from. It will be passed in the form:
  - 7.3
  - 7.4
  - 8.5
  - etc...
  Different action may be taken depending on the version passed. As of 8.6E, lettered releases are also incorporated. Currently, 7.3x is the earliest version supported.
- DEFINE INPUT PARAMETER tver AS CHARACTER . This parameter is used to determine the version of MFG/PRO the conversion is converting to. It will be passed in the form:
  - 8.5
  - 8.6
  - 8.6F
  - etc...
  Different action may be taken depending on the version passed.
- For batch background versus normal foreground:
  a) DEFINE INPUT PARAMETER howrun AS CHARACTER . This parameter will define whether a batch background run was requested or a normal foreground run was requested.
  b) Set howrun to B for background, and to F for foreground. This parameter will determine how the output is processed.
  Include two forms of output, one using a straight display statement, one using a stream for background runs. This is not currently used. There is no background run of target version programs.
- DEFINE INPUT PARAMETER logname AS CHARACTER FORMAT "x(100)" NO-UNDO . This following parameter
will only be used if a background run was requested. It will contain a path and file name for the output stream to receive output.
For example:

DEFINE STREAM foo.
OUTPUT STREAM foo TO VALUE (logname).
DISPLAY STREAM foo cnt WITH FRAME stframe.

Variables

The following variables must be defined.

- DEFINE THE VARIABLE stopvar AS LOGICAL NO-UNDO .

This variable is a hook the Conversion Shell will set. Test the value of this variable before the end of a loop to see if the user has requested a stop of the program and process the stop accordingly.

- DEFINE THE VARIABLE cnt AS DECIMAL NO-UNDO .

Frames

The following frame must be defined.

DEFINE FRAME stframe SKIP "records processed " cnt WITH 1 DOWN NO-LABELS.

Standard Code

See the target version program template below.

Transaction Scoping and Rerunability

First some background. In the past there have been some problems with very large databases, both under Progress and Oracle, with transaction scoping. For the most part the conversion programs each created one huge transaction which is either completed as a whole or completely rolled back if something goes wrong. When dealing with large databases we ran into problems with the bi file, or the lbi file or the Oracle rollback segments growing too large and causing conversion programs to abort rather awkwardly. In order to address these issues we have changed the transaction scoping so that transactions are committed in groups of 10000 records. This alleviates the problems with the bi files and the rollback segments but it creates another problem. It is now possible for a conversion program to fail for some reason in the middle of a conversion with some records converted and others not. This is bad. We therefore created a new requirement that the conversion programs be rerunnable. That is, it must be possible to run the conversion again in the case where it does not complete and have the conversion finish the job it started without damaging the data. The sample target version program below shows the recommended implementation of these features.

User Input

Messages

Messages may NOT cause a pause for any reason. Do NOT use a pause statement in your code.
Do NOT use any standard MFG/PRO messages. Do NOT use an alert-box or any type of dialog box in your code except in the following case.

Getting user input

User programs must NOT block for user input. All user input must be entered up front from the Conversion Shell.
The questions are specified in ui(xx).ini for each appropriate source release in the following format:
program-name,question-name,question,run-prog

For example:
85p5,q1,Process sales orders,none, or validation program name

Either put 'none' or, for example, the validation procedure, 85p5q1.p.
If a program name is supplied, the shell will run that program and pass it the answer the user typed in as a CHARACTER input parameter.

Display any message to the user in an alert box if their answer is incorrect.

Validation programs must have two parameters

DEFINE INPUT PARAMETER user-answer AS CHARACTER FORMAT "x(70)".
DEFINE OUTPUT PARAMETER user-ok AS LOGICAL .

SET user-ok TO FALSE . /* to force the user to re-answer the question */

Retrieving answers at run time:
/* parse input as follows */
run getNameValue in main
{input "[environment]",
 input "q1", /* the name of the question in the ui file */
 output ans-value,
 output l-notfound). /* if l-notfound is true, ans-value will be "". */
if l-notfound then do:
   ans-value = default-value.
end.

Special dump procedures

Special dumps will follow a different syntax than listed above.

Parameters

Special dump programs have one input parameter.

Shared Variables

There are two shared variables defined for use by special dump programs:

Sample Program

See the sample program below.

Creating Conversion Programs

Adding a Target Version Program to MFG/CONV (eB)

Get the latest MFG/UTIL and MFG/CONV environment from the Continuus mfgutil, mfgusrc, and conv projects.

Put your program in your local conv/us/src and xcode it into your local conv/us/xrc.

Edit tvmenu.ini to add your program’s name to all of the appropriate sections.

For each prior release, determine if your program needs to be run.

If your program needs to be run for a release, add its name to the section of tvmenu.ini for that release.

If your program depends on any other program running prior to your program make sure it is in the list after that program.

If your program has other programs that are dependent on changes that your program makes to the data, make sure that your program runs prior to the other program.

For each of the releases determined in the previous step, edit the appropriate ui(xx).ini to add any questions that are necessary.

Test your conversion program by running it from MFG/UTIL in your local environment.

Create and/or select an appropriate task for adding your changes to Continuus.

When your program passes all the tests, check out tvmenu.ini and all of the ui files that you changed (Continuus 9.1).

Add your plain text program to the conv project in us/src (9.1). Add the encrypted version to the conv project in us/xrc.

Check in the selected task.

Adding a Target Version Program to MFG/CONV (eB2)
Get the latest MFG/UTIL and MFG/CONV environment from the sysengr project of the sysengr_repository of CVS. See the Dev Systems CVS FAQs for details.

Put your program in your local conv/us/src and xcode it into your local conv/us/xrc.

Edit tvmenu.ini to add your program’s name to all of the appropriate sections.

For each prior release, determine if your program needs to be run.

If your program needs to be run for a release, add its name to the section of tvmenu.ini for that release.

If your program depends on any other program running prior to your program make sure it is in the list after that program.

If your program has other programs that are dependent on changes that your program makes to the data, make sure that your program runs prior to the other program.

For each of the releases determined in the previous step, edit the appropriate ui(xx).ini to add any questions that are necessary.

Test your conversion program by running it from MFG/UTIL in your local environment.

When your program passes all the tests, move your changes to your CVS work area.

Move your plain text program to your CVS work area.

Commit your changes.

**Sample target version program**

Here is a sample target version program that shows the correct structure.

```c
/* tvtmpit.p - Template for conversion target version program. */
/* Copyright 1986-2002 QAD Inc., Carpinteria, CA, USA. */
/* All rights reserved worldwide. This is an unpublished work. */
/* $Revision: $ */
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/* If this returns true then the conversion was a success. */
/*
*/
/* There are no issues with running this program multiple times; however this */
/* should not be necessary. */
/*
*/
/*------------------------ TOKENS ------------------------------------*/
/*V8:ConvertMode=NoConvert
*/
/*-----------------------------------------------------------------------------*/
/*--------------------------- REVISION HISTORY -------------------------------*/
/* Revision: 9.2 CREATED BY: Brian Smith DATE: 05/01/02 */
/*-----------------------------------------------------------------------------*/
define input parameter main as widget-handle.
define input parameter basever as character no-undo.
define input parameter targetver as character no-undo.
define input parameter howrun as character no-undo.
define input parameter logname as character no-undo format "x(100)".

define variable cnt as integer no-undo format "<<<,<<<,<<9";
define variable stopvar as logical no-undo.
define variable transcnt as integer no-undo.

/* Define your query, use fields option to list only the fields you need. */
define query xxx-select for xxx_mstr fields (xxx_inv_nbr xxx_ar_sub xxx_ar_acct ) scrolling.
define frame stframe
skip
"records processed " cnt
with 1 down no-labels.

pause 0 before-hide.
view frame stframe.

/* CHECK FOR APPROPRIATE VERSION (CONVERTING UP TO 9.2, NOT ABOVE) */
if basever > "9.1"
then do:
run stopok.
return.
end. /* if basever > "9.1" */

/* Display a startup message. */
run dispmes in main ("xxx Conversion in Process ").

cnt = 0.
/* Open the query no-lock with an appropriate where clause. */
open query xxx-select for each xxx_mstr where xxx_inv_nbr <> "" no-lock.
trans-loop:
/* Set up a transaction scope for the updates. */
repeat transaction:
  /* Update 1000 records, or as many as are left, within one
transaction. */
  do transcnt = 1 to 1000 on endkey undo, leave:
      /* Get the next record to process from the query. */
      get next xxx-select exclusive-lock.
      if query-off-end("xxx-select") then do:
          /* Oops, were done. (In a good way.) */
          leave trans-loop.
      end. /* if query-off-end */
      /* Insert conversion logic here. Perhaps something like this
sample.
      if xxx_ar_sub = ""
then
      assign
        xxx_ar_sub = parse_subacct (xxx_ar_acct)
        xxx_ar_acct = parse_account (xxx_ar_acct).
      /* Count and display every now and then. */
      run cnt-incr.
      /* See if someone is trying to stop us. */
      process events.
      if stopvar then do:
          run localcleanup.
          leave trans-loop.
      end. /* if stopvar */
      end. /* do transcnt = 1 to 1000 */
  end. /* repeat transaction (trans-loop) */
  /* Close the query. */
  close query xxx-select.
  /* Display appropriate results. */
  if stopvar then do:
      run dispmes in main ("xxx Conversion Stopped.").
      run dispmes in main (string(cnt - transcnt, "Records processed:
>>>,>>>,>>>9" )).
  end. /* if stopvar */
  else do:
      run dispmes in main ("xxx Conversion Completed.").
      run dispmes in main (string(cnt, "Records processed: >>>,>>>,>>>9" )).
      run stopok.
  end. /* else if stopvar */
return.

/*============================================================================*/
/*============================= INTERNAL PROCEDURES
=============================*/
/*============================================================================*/
/*------------------------------------------------------------------------------
Purpose   : Increments cnt and displays it in stframe every 100 records.
Notes     :
History   :
------------------------------------------------------------------------------*/
PROCEDURE cnt-incr:
  cnt = cnt + 1.
  if cnt modulo 100 = 0
     display cnt with frame stframe.
end PROCEDURE. /* cnt-incr */

/*============================================================================*/
/*---------------------- INTERNAL PROCEDURES----------------------------------*/
/*============================================================================*/

Purpose : Called upon successful completion of conversion. Notifies
shell
Sample special dump program

Here is a sample special dump program that shows the correct form:

```/* svabddet.p - Pre-Conversion Dump Routine for Abd_det */
/* Copyright 1986-2002 QAD Inc., Carpinteria, CA, USA. */
/* All rights reserved worldwide. This is an unpublished work. */

/* Description: This program does a special dump of Abd_det */
/* Field sequence 310 in version 7.4 (abd_udc4) is replaced with field */
/* abd_prior_ast, and field abd_udc4 is moved to field sequence 440. If the */
/* abd_udc4 value is ?, then it is reset to 0, otherwise the original value */
/* is retained. Also, a new field (abd_sched_depr) is added as sequence */
/* 430. */

/* */
/* */
/* Tokens */
-- ----------------------------------------------
/*V8:ConvertMode=NoConvert */
/*V8:RunMode=Character,Windows */
```
define input parameter main as handle no-undo.

define shared variable recount as integer.

define variable xxabd_prior_ast as decimal decimals 10 initial 0
    format "->>,>>,>>,>>,>>9.99<"
    label "Prior Depreciation--Asset Life"
    no-undo.

define variable xxabd_sched_depr as decimal decimals 10 initial 0
    format "->>,>>,>>,>>,>>9.99<"
    label "Scheduled Depreciation--Annual"
    no-undo.

define shared variable dumpdir as character format "x(80)" no-undo.
define shared variable offset as character format "x(4)" no-undo.

define variable putit as character format "x(100)" no-undo.
define variable varudec4 like abd_udec4 no-undo.
define variable inplace as logical no-undo.

define shared stream d.

run isinplace in main
    (output inplace).

putit = dumpdir + "/abd_det.d" .

session:year-offset = integer(offset).

output stream d to value(putit) .

for each abd_det no-lock:
    varudec4 = if abd_udec4 = ? then 0
                else abd_udec4.

    export stream d
        abd_book
        abd_asset
        abd_type
        abd_active
        abd_cost
        abd_curr_cost
        abd_ex_rate
        abd_ent_ex
        abd_life_yr
        abd_life_mnth
        abd_rem_yr
        abd_rem_mnth
        abd_method
        abd_salvage
        abd_dtd
        abd_ytd
The Conversion Shell Code

Due to the major changes in structure, only the eB and eB2 conversions will be explained in this document. All the eB and eB2 conversion programs reside in a directory structure under the $STGOBJ directory called the conv directory. (For example, at the UNIX prompt, with the environment set to 91, change directory to $STGOBJ/conv.) Be sure that your environment variable is set correctly in order to use these files (contact Development Systems on this issue).

Directory Structure

The structure of the conv directory is as follows:

conv  Conversion Shell, UNIX scripts, NT batch files, ini files, lst files and text files.
conv/us/xrc Encrypted conversion program files.
conv/us/src Source files of the conversion programs.
**conv/triggers** Special no-op triggers for running target version conversions against the target database. This is necessary as when certain conversion programs are run, a trigger will be fired automatically for certain tables or fields. Since the actual trigger is not needed, a no-op trigger is created for that conversion program.

**conv/dmprocs** Supplied dump procedures.

**conv/inplace** Schema, sql code and documentation for the “in place” conversions are in these directories.

### OS (Non-Progress) Files Supplied

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mfgconv.ini</td>
<td>Text file that stores values needed for the conversions.</td>
</tr>
<tr>
<td>Conv</td>
<td>Sample script to start the conversion shell. This script is not normally needed because the conversion shell is started from MFG/UTIL.</td>
</tr>
<tr>
<td>conv.bat</td>
<td>Sample batch file to start the conversion shell. This batch file is not normally needed because the conversion shell is started from MFG/UTIL.</td>
</tr>
<tr>
<td>admlist.txt</td>
<td>This is a list of the tables in the admin database. It is used by load program ld91adm.p as the list of tables to look for in the data directory.</td>
</tr>
<tr>
<td>cfglist.txt</td>
<td>This is a list of the tables in the cfg database. It is used by load program ld91cfg.p as the list of tables to look for in the data directory.</td>
</tr>
<tr>
<td>ecglist.txt</td>
<td>This is a list of the tables associated with Ecommerce. It is used by load program ld91ecg.p as the list of tables to look for in the data directory.</td>
</tr>
<tr>
<td>guilist.txt</td>
<td>This is a list of the tables in the gui database. It is used by load program ld91gui.p as the list of tables to look for in the data directory.</td>
</tr>
<tr>
<td>qadlist.txt</td>
<td>This is a list of the tables in the main database. It is used by Oracle load program ld91qad.p as the list of tables to look for in the data directory.</td>
</tr>
<tr>
<td>runconv.ksh</td>
<td>This script run the background dumps when the conversion shell is started by mfgutil.</td>
</tr>
<tr>
<td>runcconv.ksh</td>
<td>This script run the background dumps when the conversion shell is started stand-alone from the conv directory instead of from mfgutil.</td>
</tr>
<tr>
<td>nodump.lst</td>
<td>Used when converting from 7.3 or 7.4. A list of tables that are not dumped using the regular MFG/PRO dump routines. Some of these tables do not exist in the target version, some of them are replaced with standard data in the target version, and some of them have special dump programs.</td>
</tr>
<tr>
<td>nodump85.lst</td>
<td>The nodump list used when converting from 8.5.</td>
</tr>
<tr>
<td>nodump86.lst</td>
<td>The nodump list used when converting from 8.6.</td>
</tr>
<tr>
<td>nodump86e.lst</td>
<td>The nodump list used when converting from 8.6e.</td>
</tr>
<tr>
<td>nodump90.lst</td>
<td>The nodump list used when converting from 9.0.</td>
</tr>
<tr>
<td>nodump91.lst</td>
<td>The nodump list used when converting from 9.1(eB).</td>
</tr>
<tr>
<td>Svfile</td>
<td>A list of the file names, file dump names and the background stream numbers for all the files to be dumped from the source database (7.3/7.4).</td>
</tr>
<tr>
<td>svfile.85</td>
<td>A list of the file names, file dump names and the background stream numbers for all the files to be dumped from the source database (8.5).</td>
</tr>
<tr>
<td>svfile.86</td>
<td>A list of the file names, file dump names and the background stream numbers for all the files to be dumped from the source database (8.6).</td>
</tr>
<tr>
<td>svfile.86e</td>
<td>A list of the file names, file dump names and the background stream numbers for all the files to be dumped from the source database (8.6e).</td>
</tr>
<tr>
<td>svfile.90</td>
<td>list of the file names, file dump names and the background stream numbers for all the files to be dumped from the source database (9.0).</td>
</tr>
<tr>
<td>svfile.91</td>
<td>A list of the file names, file dump names and the background stream numbers for all the files to be dumped from the source database (9.1(eB)).</td>
</tr>
<tr>
<td>tvmenu.ini</td>
<td>A list of all target version programs for each destination release.</td>
</tr>
</tbody>
</table>
uibase.ini  The text file containing the set of questions for the user that are common to all source versions.

ui.ini  The text file containing the questions that the user will be asked in the initialization of the Conversion Shell’s mfgconv.ini file for source version 7.3/7.4 databases.

ui85.ini  The text file containing the questions that the user will be asked in the initialization of the Conversion Shell’s mfgconv.ini file for source version 8.5 databases.

ui86.ini  The text file containing the questions that the user will be asked in the initialization of the Conversion Shell’s mfgconv.ini file for source version 8.6 or 8.6e databases.

ui90.ini  The text file containing the questions that the user will be asked in the initialization of the Conversion Shell’s mfgconv.ini file for source version 9.0 databases.

ui91.ini  The text file containing the questions that the user will be asked in the initialization of the Conversion Shell’s mfgconv.ini file for source version 9.1 databases.

version.cnv  A version file for the conv directory.

**Progress Code**

**Conversion Shell**

<table>
<thead>
<tr>
<th>Program Name</th>
<th>Description</th>
<th>Programs run or includes</th>
</tr>
</thead>
<tbody>
<tr>
<td>mfgconv.p</td>
<td>mfgconv.p is the main conversion shell program.</td>
<td>{initmptb.i} {iniuitt.i} {initmpl1.i} {svtmpptb.i} {svndump.i} schema.p ckschema.p svmdump.p getoranm.p runcmd.p tablelog.p</td>
</tr>
<tr>
<td>Bgconv.p</td>
<td>Version of mfgconv.p for doing background dumps.</td>
<td>same as above</td>
</tr>
<tr>
<td>runconv.p</td>
<td>A stand alone version of the shell for running individual source version or target version programs.</td>
<td>same as above</td>
</tr>
<tr>
<td>Svp3.p</td>
<td>Dumps the source database tables (foreground and background).</td>
<td>{initmptb.i} {svtmpptb.i} {svndump.i}</td>
</tr>
<tr>
<td>Svp4.p</td>
<td>Generates the bulkload description files.</td>
<td>svtmpptb.i writefd.p</td>
</tr>
<tr>
<td>Svp5.p</td>
<td>Runs the Progress bulkload.</td>
<td>runcmd.p</td>
</tr>
<tr>
<td>Svp6.p</td>
<td>Runs the Progress re-index.</td>
<td>runcmd.p</td>
</tr>
<tr>
<td>svmdump.p</td>
<td>Allows the user to choose the background stream if performing a full background dump and load conversion.</td>
<td></td>
</tr>
<tr>
<td>chkttable.p</td>
<td>Indicates whether a table is present in a database.</td>
<td></td>
</tr>
<tr>
<td>ckschema.p</td>
<td>Checks the schema of the source database.</td>
<td></td>
</tr>
<tr>
<td>Schema.p</td>
<td>Presents user option of choosing release when there is disagreement between mfgconv.ini and database as to the release of the database.</td>
<td></td>
</tr>
<tr>
<td>getoranm.p</td>
<td>Gets the name of the Oracle database when the connected database is a schema holder.</td>
<td></td>
</tr>
<tr>
<td>Runcmd.p</td>
<td>Part of mfgutil. Not in the conv project or directory but used by conversions. Runs operating system commands. Found in the mfgutil/mfgusrc directory in the CVS repository.</td>
<td></td>
</tr>
<tr>
<td>Program Name</td>
<td>Description</td>
<td>Programs run or includes</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>tablelog.p</td>
<td>Returns the list of connected logical database names that contain the specified table.</td>
<td></td>
</tr>
<tr>
<td>Writefd.p</td>
<td>Write a bulkload description file (.fd) for the dumped tables.</td>
<td></td>
</tr>
<tr>
<td>ld91adm.p</td>
<td>Load program for admin DB.</td>
<td></td>
</tr>
<tr>
<td>ld91all.p</td>
<td>Oracle load program for all tables.</td>
<td></td>
</tr>
<tr>
<td>ld91cfg.p</td>
<td>Load program for cfg DB.</td>
<td></td>
</tr>
<tr>
<td>ld91ecg.p</td>
<td>Load program for Ecommerce tables.</td>
<td></td>
</tr>
<tr>
<td>ld91gui.p</td>
<td>Load program for gui DB.</td>
<td></td>
</tr>
<tr>
<td>ld91quad.p</td>
<td>Oracle load program for main DB.</td>
<td></td>
</tr>
<tr>
<td>convenv.p (eB2)</td>
<td>Dialog for conversion environment variables.</td>
<td></td>
</tr>
<tr>
<td>srcdbinf.p (eB2)</td>
<td>Dialog for source version database information.</td>
<td></td>
</tr>
<tr>
<td>Srcdbin1.p (eB2)</td>
<td>Dialog for extra progress source database for 7.4 oracle conversion.</td>
<td></td>
</tr>
<tr>
<td>Trgdbin1.p (eB2)</td>
<td>Dialog for target version qaddb database.</td>
<td></td>
</tr>
<tr>
<td>Trgdbin2.p (eB2)</td>
<td>Dialog for target version admin database.</td>
<td></td>
</tr>
</tbody>
</table>

**Source Version Programs**

<table>
<thead>
<tr>
<th>Program Name</th>
<th>Description</th>
<th>Programs run or includes</th>
</tr>
</thead>
<tbody>
<tr>
<td>svcont02.p</td>
<td>Source Conversion program for overlapping contracts</td>
<td></td>
</tr>
<tr>
<td>Svfscon1.p</td>
<td>Pre-Conversion SSM Contracts Report</td>
<td>svcont02.p</td>
</tr>
<tr>
<td>Svfscon2.p</td>
<td>Pre-Conversion Establish Blank Contracts</td>
<td></td>
</tr>
<tr>
<td>Svfscon3.p</td>
<td>Pre-Conversion SSM Installed Base Report</td>
<td>svcont02.p</td>
</tr>
<tr>
<td>svindrep.p</td>
<td>Conversion Report for lngd_det</td>
<td></td>
</tr>
<tr>
<td>svngrep.p</td>
<td>Conversion Report for lng_mstr</td>
<td></td>
</tr>
<tr>
<td>svmndrep.p</td>
<td>Conversion Report for mnd_det</td>
<td></td>
</tr>
<tr>
<td>svmsgrep.p</td>
<td>Conversion Report for msg_mstr</td>
<td></td>
</tr>
<tr>
<td>svrepv.p</td>
<td>Report Viewer for the Conversion Shell</td>
<td></td>
</tr>
</tbody>
</table>

**Special Dump Programs Supplied by the Project Teams**

<table>
<thead>
<tr>
<th>Program Name</th>
<th>Description – Project Team supplied programs</th>
<th>Programs run or includes</th>
</tr>
</thead>
<tbody>
<tr>
<td>svabddet.p</td>
<td>Pre-conversion dump program for table abd_det</td>
<td></td>
</tr>
<tr>
<td>svcmmstr.p</td>
<td>Pre-conversion dump program for table cm_mstr</td>
<td></td>
</tr>
<tr>
<td>svfa.p</td>
<td>Pre-Conversion Special Dump for Fixed Assets</td>
<td></td>
</tr>
<tr>
<td>Svfplan.p</td>
<td>Pre-conversion dump program for table flp_plan</td>
<td></td>
</tr>
<tr>
<td>svfpdcos.p</td>
<td>Pre-conversion dump program for table fpd_cost</td>
<td></td>
</tr>
<tr>
<td>svfpplan.p</td>
<td>Pre-conversion dump program for table fp_plan</td>
<td></td>
</tr>
<tr>
<td>svfrqmst.p</td>
<td>Pre-conversion dump program for table frq_mstr</td>
<td></td>
</tr>
<tr>
<td>Program Name</td>
<td>Description</td>
<td>Programs run or includes</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>tv86gs1.p</td>
<td>Global Shipping conversion</td>
<td>tvgsa1a.p</td>
</tr>
<tr>
<td>tv86gs1a.p</td>
<td>Global Shipping conversion sub program</td>
<td></td>
</tr>
<tr>
<td>tvcdr1.p</td>
<td>Currency Dependent Rounding</td>
<td>initmptb.i, gmfcctl.i, tvcdr1a.p, tvcdr2.p, tvcdr3.p</td>
</tr>
<tr>
<td>tvcdr1a.p</td>
<td>Currency Dependent Rounding sub program</td>
<td></td>
</tr>
<tr>
<td>tvcdr2.p</td>
<td>Currency Dependent Rounding sub program</td>
<td></td>
</tr>
<tr>
<td>tvcdr3.p</td>
<td>Currency Dependent Rounding sub program</td>
<td></td>
</tr>
<tr>
<td>Tvcmcnd85.p</td>
<td>Corporate Commodities Code Conversion</td>
<td></td>
</tr>
<tr>
<td>tvcomcd.p</td>
<td>Corporate Commodities Code Conversion</td>
<td></td>
</tr>
<tr>
<td>tvcuex.p</td>
<td>Triangulation data conversion</td>
<td>triac.i, trivac.i</td>
</tr>
<tr>
<td>tvcupcconv.p</td>
<td>Cum Ship Reset Conversion Program</td>
<td></td>
</tr>
<tr>
<td>tveas.p</td>
<td>Extended Account Structure Conversion Program</td>
<td></td>
</tr>
<tr>
<td>tvecg.p</td>
<td>ECommerce Conversion post-processing program MFG/PRO 9.0</td>
<td></td>
</tr>
<tr>
<td>tvecg92.p</td>
<td>Ecommerce post processing program for 9.2</td>
<td></td>
</tr>
<tr>
<td>tvers.p</td>
<td>Evaluated Receipts Settlement (ERS) Conversion. Creates records for every site and supplier in a database that set the ERS option on all purchase orders to 1, this disallows ERS processing.</td>
<td>initmptb.i</td>
</tr>
<tr>
<td>tvfscont.p</td>
<td>Field Service conversion sub program</td>
<td></td>
</tr>
<tr>
<td>tvglo91.p</td>
<td>Conversion Routine For arc/apc_gl_sum to arc/apc_sum_lvl</td>
<td></td>
</tr>
<tr>
<td>tvgtm.p</td>
<td>Global Tax Management conversion</td>
<td></td>
</tr>
<tr>
<td>tvlsc91</td>
<td>Linked-Site Costing for 9.2</td>
<td></td>
</tr>
<tr>
<td>tvmrp.p</td>
<td></td>
<td></td>
</tr>
<tr>
<td>File</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>tvmrp1.p</td>
<td>mrp for 9.2</td>
<td></td>
</tr>
<tr>
<td>tvp2.p</td>
<td>Fix RCPT-STAT / W.Orders. Conversion moves various fields from the work file to new locations in the database schema to enable the operation of Compliance programs.</td>
<td></td>
</tr>
<tr>
<td>tvp3.p</td>
<td>Mix Variance on Product Lines / W.orders</td>
<td></td>
</tr>
<tr>
<td>tvp4.p</td>
<td>Price List Conversion</td>
<td></td>
</tr>
<tr>
<td>tvp5.p</td>
<td>Price List History Conversion</td>
<td></td>
</tr>
<tr>
<td>tvp6.p</td>
<td>Menu Update to qad_wkfl</td>
<td></td>
</tr>
<tr>
<td>tvp7.p</td>
<td>Bank Detail Conversion</td>
<td></td>
</tr>
<tr>
<td>tvp8.p</td>
<td>Multi Language. Ensures that default fields in the ‘us’ directory are populated if more than one language is used.</td>
<td></td>
</tr>
<tr>
<td>tvp9.p</td>
<td>Field Service conversion</td>
<td></td>
</tr>
<tr>
<td>tvpcc0.p</td>
<td>PCC Conversion. Moves Engineering Change data to new locations in the database. No data is changed, just relocated.</td>
<td></td>
</tr>
<tr>
<td>tvqadvph.p</td>
<td>qad_wkfl To vph_hist Conversion Program</td>
<td></td>
</tr>
<tr>
<td>Tvsaddet.p</td>
<td>SSM for 9.2</td>
<td></td>
</tr>
<tr>
<td>Tvsmast.p</td>
<td>SSM for 9.2</td>
<td></td>
</tr>
<tr>
<td>tvsite.p</td>
<td>Copy Site To Supplier Performance Conversion Program</td>
<td></td>
</tr>
<tr>
<td>tvssm2.p</td>
<td>SSM for 9.2</td>
<td></td>
</tr>
<tr>
<td>tvtri.p</td>
<td>Triangulation data conversion</td>
<td></td>
</tr>
<tr>
<td>tvusr92</td>
<td>User count audits for 9.2</td>
<td></td>
</tr>
</tbody>
</table>

Ini File Structures

The mfgconv.ini file structure

The [Versions] Section

This section consists of a release string and the name of a table whose absence will identify that release.

```
# Supported database versions
# Each version except 7.3 and the latest version has a table name that, # if it cannot be found, means that the database being checked is the # named version. The tables must be checked in the order given. # The code to do this checking is in schema.p.
[Versions]
7.3
7.4=acm_mstr
8.5=absr_det
8.6=exru_usage
8.6e=aprm_mstr
9.0=abss_det
eB
```

The [uifile] Section
This section consists of a release string and the name of the ui file that has the questions for that release. There is also a false release string of “base” that is for use before the release is known.

```ini
# Version to ui file name correlation.
[uifile]
base=ui.base.ini
7.3=ui.ini
7.4=ui.ini
8.5=ui85.ini
8.6=ui86.ini
8.6e=ui86.ini
9.0=ui90.ini
```

**The [svfile] Section**

This section consists of a release string and the name of the file that has the source version program information for that release.

```ini
# Version to svfile name correlation.
[svfile]
7.3=svfile
7.4=svfile
8.5=svfile.85
8.6=svfile.86
8.6e=svfile.86e
9.0=svfile.90
```

**The [nodump] Section**

This section consists of a release string and the name of the file that has the no dump information for that release.

```ini
# Version to nodump name correlation.
[nodump]
7.3=nodump.lst
7.4=nodump.lst
8.5=nodump85.lst
8.6=nodump86.lst
8.6e=nodump86e.lst
9.0=nodump90.lst
```

**The [tvprogs] Section**

This section consists of a release string and the name of the section in tvmenu.ini used for that release.

```ini
# Version to tvmenu.ini section name correlation.
[tvprogs]
7.3=tvprogs74
7.4=tvprogs74
8.5=tvprogs85
8.6=tvprogs86
8.6e=tvprogs86e
9.0=tvprogs90
eB=tvprogs91
```

**The [specialdumps] Section**

This section consists of a release string and the name of the section in mfgconv.ini used for the special dumps for that release.
The **[special174]** Section

This section consists of a comma-separated list of tables and the program to run to accomplish the special dumps for those tables. This list is for release 7.3 and 7.4.

```
# Version to special dump section name correlation.
[specialdumps]
7.3=special174
7.4=special174
8.5=special85
8.6=special86
8.6e=special86e
9.0=special90
9.1=special91
```

The **[special174]** Section

This section consists of a comma-separated list of tables and the program to run to accomplish the special dumps for those tables. This list is for release 7.3 and 7.4.

```
# Special dump tables and programs for pre-85.
[special174]
itm_det=svitmdet.p
sal_mstr=svsalmstr.p
cm_mstr=svcmstr.p
abd_det=svabddet.p
sm_mstr=svsmstr.p
pald_det=svpaldet.p
pal_mstr=svpalmstr.p
lnd_det=svlndet.p
fpd_cost=svfpdcost.p
flp_plan=svflplan.p
frq_mstr,rq_mstr=svfrqmst.p
fp_plan,fp1_plan,fp2_plan,fp3_plan=svfpplan.p
fslp_plan,fsp1_plan,fsp2_plan,fsp3_plan=svfsplan.p
cu_mstr,exr_rate=svtri74.p
```

The **[special 85]** Section

This section consists of a comma-separated list of tables and the special dump program that will dump those tables. This list is for release 8.5.

```
# Special dump tables and programs for 85.
[special85]
cu_mstr,exr_rate=svtri85.p
```

There are additional **[specialxx]** sections for each additional release. See the mfgconv.ini file.

The **[FixedAssets]** Section

This section consists of a comma-separated list of tables and the special dump program that will dump those tables. This list is for fixed assets.

```
# Special dump tables and programs for fixed assets.
[FixedAssets]
dpr_mstr, dbk_mstr, fal_mstr, cls_mstr, ast_mstr, abd_det, dprd_det, accd_det=svfa.p
```

The **[NoBulk]** Section

This section lists tables that are excluded from the bulk load.

```
# List of tables to exclude from the bulk load, i.e. specially formatted
tables
# that are loaded by tv programs.
[NoBulk]
cu_mstr
```

The **[Environment]** Section

The section **[environment]** stores the answers to the questions asked in the question frame. Each variable is
mapped to the ui.ini file as well.

This section is initialized by the shell with the user’s answers.

**The [svprograms] Section**

This section is the list of programs to run to accomplish the source version dumps.

```
# Program run against the Source Database for Progress
[svprograms]
svp3.p
```

**The [OracleProgs] Section**

This section is the list of programs to run to accomplish the source version dumps for Oracle.

```
# Section to run the Programs for Oracle
[OracleProgs]
svp3.p
```

**The [bulkload] Section**

```
# Program to run the Progress Bulkload Program
[bulkload]
svp5.p
```

**The [index] Section**

```
# Program to run the Progress Index Rebuild Program
[index]
svp6.p
```

**The [Programs] Section**

The programs get a status of started or completed. If status remains started, it means the program did not complete. This section is filled in by the shell as programs are run.

**The ui file structure**

This file is used by the question frame to query the user for conversion input. The basic structure of the file is as follows (commas are used as delimiters):

- Reference information that will be displayed below the question.
- The variable in the mfgconv.ini file to store the answer in.
- The text the user will see on the screen.
- A validation program supplied by the project team to validate the answer or “none” if no validation is necessary.

There are the following versions of this file:

- ui.ini
- ui85.ini
- ui86.ini
- ui90.ini
- ui91.ini

**The svfile file structure**

This file consists of a comma-separated list of:

- table-name, dump-name, queue, dumped-flag.

The original file does not have the dumped-flag in it. The dumped-flag field is generated by the shell as necessary.

The original file also has the queue set to 6 for all files.

There are the following versions of this file:

- svfile
- svfile.85
- svfile.86
- svfile.86e
- svfile.90
- svfile.91
The tvmenu.ini structure

This file is used to display the menu items for the target version programs. It also ensures that the programs run in the correct order and that the required programs are run. There is a section for each release (see the svfile section of mfgconv.ini above). Within each section, each entry consists of:
Filename, Sequence Order, Description, Required.

The nodump file structure

This file consists of a list of table names that should not be dumped. There are the following versions of this file:
nodump.lst
nodump85.lst
nodump86.lst
nodmp86e.lst
nodump90.lst

History of Versions of the Conversions

The following versions of the conversions exist.

7.3-7.4 to 8.5 (UNIX) (8.5 Conversions)

7.3-7.4 to 8.5 (NT)

These two versions are exactly the same, but have a few programs pre-compiled differently for the release. This was done because the wrong pre-compiled versions were showing up on the release.

7.3-7.4-8.5 to 8.6 (8.6 Conversions)

One version is used for both NT and UNIX as all the initial programs are supplied as source and encrypted files making pre-compilation unnecessary.

7.3-7.4-8.5-8.6 to 9.0 (UNIX and NT) (9.0 Conversions)

Pretty much the same as 8.6.

To eB and above (eB and eB2 Conversions)

The main differences between the 8.5, 8.6, or 9.0 conversions are as follows:

- The 8.5 conversion assumes the following programs are delivered pre-compiled on the release: mfgconv.p, mfgconvnt.p, sv0.p, svp0.p, and tvp0.p. The 8.6 conversions do not require pre-compiles.
- The 8.5 conversions require the end user to enter the version of their source database. The 8.6 conversions determine this automatically.
- The 8.5 conversions allow (though incorrectly) an end user to run target programs against the source database. The 8.6C and 8.6D conversions enable and disable the menu structure accordingly, however, this is no longer true for 8.6E or beyond.
- The eB conversions are not compiled at all. All of the external batch files and scripts that needed to be tailored have been eliminated.

Rationale

Use of Source Version (sv) conversion programs:

'sv' conversion programs are used in cases where the structure of the source DB schema is different from the target DB schema. In these cases, a simple 'dump and load' of the DB is not possible. Typical scenarios requiring 'sv' conversions include:

- a table in the source version schema is being obsoleted in the target version, and data in the source version needs to be relocated to new table(s) in the target version
- field(s) in the source version are being redefined in the target version schema, requiring data in the source version to be 'reformatted' so that it can be imported/loaded into the target version schema (i.e., a field in the source version has a "CHARACTER" format and in the target version the field has an "INTEGER" format).

Use of Target Version (sv) conversion programs:

'tv' conversion programs are most commonly used in the following scenarios:
• a new field is being introduced in the target version. In the source version, the comparable data was maintained in a "QAD Reserved" field, or qad_wkfl field. The tv conversion program moves the data from the source version schema location to the new field(s).
• a new table is being introduced in the target version. Data in one or more source version tables is 'moved' to the new table.
• a new field is added that must be populated with data from another field in another table.

See the attached document for examples and templates.

Examples

See the attached document for examples and templates.

See Also

N/A
DST-0005 Developing with OID fields (eB3 and up)

Summary

This document contains an overview and guidelines for developing with OIDs. In eB3 and above, all tables must contain an OID field. In all qaddb tables, this OID field uniquely identifies each record. The attached Word document provides an overview of the design and coding considerations when developing with OIDs. References to specific standards are also included.

<table>
<thead>
<tr>
<th>ID</th>
<th>DST-0005</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applicability</td>
<td>All development activities for eB3 and above.</td>
</tr>
</tbody>
</table>

Description

Introduction

In eB3, a unique record identifier field, referred to as an OID for “object ID”, has been added to each operational table in the qaddb database (but not to the Financials tables). This field was originally proposed by the Medical CFR Part 11 project to facilitate the linking of audit trails and electronic signatures to MFG/PRO records. The MFG/PRO architects approved the introduction of these new fields and have added and modified several development standards to gain the most benefit from the OID fields. This document provides an overview of the design and coding considerations when developing with OIDs.

Schema Design

An OID field and corresponding unique index have been added to all existing, non-Financials qaddb tables in the eB3 baseline schema. Starting in eB3, all new operational, non-Financials schema tables added to qaddb must include the following:

- **Main OID field**: Each new table must contain an OID field named oid_<table_name> where <table_name> is the Progress table name (e.g. gl_mstr). Please refer to **STD-0050 Required Table Fields**.

- **Main OID index**: Each new table must contain a unique index consisting of just the main OID field. The value of the OID field is unique even across Domains and so the main OID index should contain only the OID field. The name of this main OID index will be exactly the same as the OID field name and will not begin with the table prefix. The index must be unique and usually will not be primary. Please refer to **STD-0327 Every table must have a unique index & a primary index (eB3 and up)**.

- **Create trigger**: Every new table must contain a create trigger to populate the main OID field. This create trigger must be defined in the schema and the corresponding .t file must use the generic include files provided in the eB3 baseline. Please refer to **STD-0326 Every qaddb table must have a create trigger (eB3 and up)**.

Any new table added to the admin or help database must contain a main OID field but should not contain a main OID index or create trigger. This is because the OID fields in these two databases have not been activated. In these databases, the OID field value will remain the default value of 0 for all records until sometime in the future when the OID fields in these databases will be activated.

When a relationship between two tables is introduced in the schema using new or existing tables, a foreign key OID field should be added to one of the tables along with a corresponding index. Please refer to **STD-0325 Use OID fields as foreign keys to relate qaddb tables (eB3 and up)**.

- **Foreign key OID field**: The main OID field for the related (e.g. parent) table should be added as a foreign key field in the dependent (e.g. child) table. The name of the foreign key OID field will match the main OID field in the related table unless additional foreign keys to the same table are needed. In this case, a meaningful suffix may be added to the field name. The attributes of the foreign key OID field will match those of the main OID field in the related table except for the description which identifies it as a foreign key.

- **Foreign key index**: An index should be added to facilitate joins between the two tables. The naming convention for the index is <prefix><foreign_key_OID_field> where <prefix> is the prefix for the table containing the index and <foreign_key_OID_field> is the name of the foreign key OID field in this table. The index will consist of the foreign key field optionally followed by additional fields. The index does not need to be unique.
Usage in Source Code

Main OID Field Population

In Database Records

The main OID field in each table is populated by a create trigger. Please refer to STD-0326 Every qaddb table must have a create trigger (eB3 and up). The logic uses generic include files and is shown below for the ad_mstr table from the adc_t source file:

```plaintext
TRIGGER PROCEDURE FOR CREATE OF ad_mstr.

{gpoidfcn.i} /* Contains 'nextOidValue' function */

{gpoidcr.i &TABLE-NAME=ad_mstr}
```

In Temp-Table Records

If a temp-table is defined “like” a database table, the OID field and unique OID index will be included and so a value must be assigned to the OID field if multiple records will exist in the temp-table. This main OID field must be populated without the convenience of a create trigger. For this situation, the nextOidValue function defined in gpoidfcn.i can be used to generate a new OID value. This is the same generic function that is called from create triggers. In the example below, two temp-table records are created and populated and are then persisted in the database through the use of a buffer-copy statement.

```plaintext
{gpoidfcn.i} /* Defines nextOidValue() function */
define temp-table tt-adcd_det like adcd_det no-undo.

create tt-adcd_det.
tt-adcd_det.oid_adcd_det = nextOidValue().
/* Additional fields are then assigned in the tt-adcd_det record. */
...
for each tt-adcd_det:
    create adcd_det.
    buffer-copy tt-adcd_det to adcd_det.
    /* The above statement creates a new adcd_det record and the */
    /* create trigger populates the OID field. But then all values from */
    /* the tt-adcd_det record are copied in including the OID value. */
    ...
end.
```

Buffer-copy Considerations
Care must be taken when using the buffer-copy statement to ensure that the correct OID value results in the target record.

When copying fields from one database record to a new record in any database table, the OID field should always be excluded from the buffer-copy statement. When the new record is created, either during the buffer-copy or in a prior create statement, the create trigger on that table will populate the main OID field. So when fields are then copied from the source record to this new target record, the OID field should not be copied so that the newly created OID value remains intact. This is no different than the considerations that would be made for fields making up any unique index.

When copying a database record to a temp-table defined “like” the database table, you must decide if the OID value from the database record should be used or whether a new value should be generated. If the temp-table record is a temporary copy of the database record that, for example, will be modified by the UI to change one or more values and then will be copied back to the same database record, then the OID value from the database record must be preserved. Use a buffer-copy statement to accomplish the copy to and from the temp-table record as shown in the sample code below.

```sql
buffer-copy adcd_det to tt-adcd_det.
/* Changes are then copied back to the database record. */
for first adcd_det where adcd_det.oid_adcd_det = tt-adcd_det.oid_adcd_det
exclusive-lock:
  buffer-copy tt-adcd_det to adcd_det.
end.
```

If the temp-table record will ultimately be used to create a new database record, then we must ensure that the new database record has a new OID value. This can be accomplished in two ways. The OID value from the database record can be copied to the temp-table record but then it must be excluded from the buffer-copy statement when buffer-copied back to a database record as shown below.

```sql
buffer-copy adcd_det to tt-adcd_det. /* OID value copied from adcd_det */
/* Changes are made to primary and other tt-adcd_det fields as needed. */
create adcd_det. /* Create trigger populates OID field */
buffer-copy tt-adcd_det to adcd_det except oid_adcd_det.
/* The above statement copied all values except for the OID value. */
```

Alternatively, a new OID value can be generated for the temp-table record and then copied to the new database record. A nextOidValue function is defined in gpoidfcn.i and is used to generate a new OID value. This is the same generic function that is used in the create triggers.

```sql
{gpoidfcn.i} /* Defines nextOidValue() function */
create tt-adcd_det.
tt-adcd_det.oid_adcd_det = nextOidValue().
buffer-copy adcd_det to tt-adcd_det except oid_adcd_det.
/* Changes are made to primary and other tt-adcd_det fields as needed. */
create adcd_det. /* Create trigger populates OID field */
buffer-copy tt-adcd_det to adcd_det.
/* The above statement copied all values including the OID value from the tt-adcd_det record. */
```

### Foreign Key OID Field Population

In the source code, the main OID field value for a new record is available immediately after the create statement. This value can then be used to populate foreign key fields in related records. Because the same field name may exist in multiple tables, each reference to an OID field must be qualified with the table name. For example:
Other Buffer Statement Considerations

In addition to the buffer-copy statement that acts on all fields in a buffer, there are other Progress statements that act on all fields of a buffer that have special OID field considerations.

**Buffer-compare**

Similar to the considerations for the buffer-copy statement, in some situations, a buffer-compare should compare the OID fields in the two records and in some cases should not. For example, if two buffers from the same table are being compared to see if both buffers point to the same record, then the OID field should be compared (don’t list in the EXCEPT field-list).

If a database record buffer is compared to a temp-table buffer defined “like” the database table to see if the record is an exact copy, then compare the OID field. This of course is assuming that the temp-table record was originally created from a buffer-copy from the database record.

If a temp-table record buffer that was originally created from a buffer-copy from a database record is compared to a database record buffer from a different database, the OID fields will most likely differ. In this case, exclude the OID field from the buffer-compare. An example of this in legacy MFG/PRO is in multi-database COP code.

**Assign and Update**

Each of these statements includes a syntax that can act on an entire buffer rather than individual fields as shown here:

```
ASSIGN { record [ EXCEPT field ... ] }
[ NO-ERROR ]
UPDATE { record [ EXCEPT field ... ] [ frame-phrase ] }
[ NO-ERROR ]
```

In general, when using these statements, the OID field should be listed in the EXCEPT field-list since the OID field will be populated when the record is created and should not be changed.

**Export and Import**

Each of these statements includes a syntax that can act on an entire buffer rather than individual fields as shown here:

```
EXPORT [ STREAM stream ] [ DELIMETER character ]
{ record [ EXCEPT field ... ] }
IMPORT [ STREAM stream ] [ DELIMETER character ]
{ record [ EXCEPT field ... ] } [ NO-ERROR ]
```

If these statements are used in archive and restore programs, then the OID field should not be listed in the EXCEPT field-list because it is important to preserve these unique record identifiers. If on the other hand, these statements are used to export records from one database and then import them to another database, the OID field would usually be excluded (listed in the EXCEPT field-list) so that the records created in the target database will have OID values generated specifically for that database.

Note: It is important that the OID field is either included or excluded in matching export and import programs for the same data. In general, if the OID field is included in the export statement (not listed in EXCEPT field-list), that data...
cannot be subsequently imported with the OID field excluded and vice-versa. There is a technical loophole that will allow this to work if the OID field happens to be the last field added to a table’s schema. This will be the case in eB3 for each table until another field is added by a project.

Technical Requirements Met by OIDs

1. A single unique value must be stored with each record in the database. This field must be indexed for fast retrieval. The Progress ROWID would have met this requirement except that the value is not retained after a dump and load. We added an OID field and unique index to each table in MFG/PRO. The unique record value will be referred to as an OID in the remaining requirements.

2. The foreign key relationships between two tables should be identified by the inclusion of the foreign table’s OID field in the table. If this field in each table has the exact same name and is indexed, convenient Progress 4GL constructs can be used for record retrieval (e.g. FOR EACH child_table OF parent_table). Third party relational database tools will also detect these table relationships from the naming convention and presence of indexes.

3. The OID value for a new record must be available in the code immediately after a create statement so that this value can be used as foreign key values in related records. This was accomplished by the use of a create trigger which also provided a non-invasive way to make this change to our massive legacy code. Specifically, our algorithm produces a decimal with the following structure:

4. When creating records in temp-tables defined "like" a database table, the unique index on the OID field requires that the OID value be populated without the convenience of the create trigger. In these cases, a nextOidValue function was provided to obtain the unique value. This required changes to legacy code.

5. The unique record value must remain unique even if records are archived and then restored many years later. We satisfied this requirement by embedding the create date in the value.

6. When copying data between database records and temp-table records, care must be taken to ensure that the correct OID values are preserved. In general, new database records require new OID values but if those new records originate as temp-table records with foreign key relationships to other temp-table records, then the OID values from the temp-table records should be preserved when those records are persisted to the database. This required changes to legacy code.

7. The unique record value should be unique across databases (in the universe) to facilitate merging databases while maintaining record uniqueness. Our OID population routine guarantees this uniqueness by embedding a db identifier (read from a control file) in the value. The identifier is based on the Progress Dynamics site id.

Examples

N/A

See Also

STD-0050 Required Table Fields
STD-0327 Every table must have a unique index & a primary index (eB3 and up)
STD-0326 Every qaddb table must have a create trigger (eB3 and up)
STD-0325 Use OID fields as foreign keys to relate qaddb tables (eB3 and up)
DST-0006 Design Considerations for Domained MFG-PRO Versions (eB2.1 and up)

Summary

The release of MFG/PRO eB2.1 introduces a new data object, Domain, into almost all MFG/PRO tables & indexes. Domain should be taken into account in the design of any new tables, and where functions are required to operate across domains.

<table>
<thead>
<tr>
<th>ID</th>
<th>DST-0006</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applicability</td>
<td>All development activities for MFG/PRO eB2.1 and up</td>
</tr>
</tbody>
</table>

Description

Introduction

The following are guidelines that should be considered for any new designs based on eB2.1 or later releases. Due to architectural changes in MFG/PRO for domain, there are new considerations that must be taken into account when designing new tables and functionality. For example, a project team must decide whether or not to add a domain field to new tables, and whether any new functionality must be able to cross domains.

Prerequisites

The Architecture Design Model and the Design Specifications for Domain should be read to provide insight into how Domain was developed. All documents are located in the “Dev Deliverables 9.2 up” Lotus Notes Database.

Overview: What is a "Domain"?

With the release of MFG/PRO eB2.1, a new data object, “Domain”, is introduced into MFG/PRO (table dom_mstr). Most of the other data objects in MFG/PRO are linked to the Domain object through a “belongs to this domain” relationship.

A field to hold the domain value has been added to the majority of the tables in the MFG/PRO schema. The field name used in each case follows the current QAD naming standards of tablePrefix_domain (e.g. cm_domain, pt_domain, etc.).

These new fields have become the first component in all indices on the tables to which they have been added, except for the main OID indices. To efficiently access the data in these tables, the domain field must form a part of any data query, and it must be populated whenever a record is created in the table.

The Domain object has been introduced in order to logically partition information in the MFG/PRO database by Domain. This object can be used to represent different legal businesses, with different control file parameters, within a single MFG/PRO database. In previous versions, such representations required the use of separate MFG/PRO databases.

This concept works by introducing the new data architecture and a new globally scoped variable: global_domain. This variable is set to a valid Domain value at the point at which a user logs in to MFG/PRO, or changes their current working Domain. From that point on, all data access statements should limit the data retrieved to the global_domain value (or a Domain value suitable for the application involved).

Database Considerations

With the eB2.1 release, a domain field was added to the majority of tables in MFG/PRO, and to all the indices of these tables. However, domain was not added to those tables where data is shared between all domains, such as languages, printers, users, menus, countries and currencies. This is typically system data.

Should new tables include domain?

From eB2.1 forward, whenever a new table is added, a decision must be made whether to include domain or not. Below there are a number of criteria which can assist in making this decision, but this list is not necessarily exhaustive, and so the question should still be asked even if none of these criteria apply.
The table holds a cost or financial value, which is only stored in base currency (e.g. Credit Limit on Customer Master (cm_mstr.cm_cr_limit)). The base currency is defined per domain, thus any table storing a value assumed to be in base currency must include domain.

The table holds an exchange rate, which relates to the base currency (e.g. Accounts Receivable Master (ar_mstr.ar_ex_rate)).

The table holds a GL Account Code, Sub-Account Code, Cost Center, or Project (e.g. Voucher Detail table (vod_acct, vod_sub, vod_cc, vod_project)). The GL Chart of Accounts is defined per domain, thus any table storing account information must include domain.

The table stores master file data that may be unique to the business (e.g. the Shop Calendar (shop_cal) contains domain because the calendar definition may be unique to a particular business unit).

The table is a control file (e.g. Inventory Control (icc_ctrl)). One of the goals of the domain project is to provide the ability to set control parameters differently by domain, thus control files should almost always include domain. On rare occasions, there may be exceptions where certain control parameters are really database specific, and not domain specific.

The table holds numbered documents (e.g. Sales Orders, Purchase Orders, Work Orders), or detail records associated with such documents (e.g. Sales Order Details, Work Order Details).

The table is directly related to another table that includes domain for one of the reasons mentioned above (holds a cost or financial value in base currency, holds an exchange rate, etc.), or the table holds a field that relates to another ‘domained’ table (e.g. Unit of Measure Master (um_mstr), where um_part relates to pt_part). An exception can be made if the table is related to the ‘domained’ table solely via a foreign key OID field.

If none of the above criteria apply to a specific table, then that table may not need domain. However, that should not be an automatic decision, and there should be a logical reason for not adding domain. Some of the criteria may include:

- The table holds ‘system’ data that should be shared across domains, as mentioned above.
- The table stores detail data for a ‘master’ table that is not domained.
- The table stores detail data for a ‘master’ table that IS domained, but the detail table is always accessed through the master table, and the detail table has a foreign key relationship with the master table that does not include the domain field. For example, if the detail table’s only relationship with the master table is through a foreign key OID field, then adding domain to the detail table is unnecessary. Another example is the exru_usage table stores exchange rate detail for several master tables (e.g. ar_mstr) that include domain, but is ALWAYS accessed via a sequence field (e.g. ar_exru_seq) on that master table. In this case, adding domain to exru_usage would be redundant, and is therefore unnecessary.

All new projects must keep the above criteria in mind when deciding whether or not to include domain on new tables.

Index considerations

For eB2.1, all tables that had a domain field added, also had the domain field added as the first component of all existing indices. For eB3 and up, the same practice should be followed with the exception of any indices containing OIDs. Since OIDs are unique within a database (and not a domain), adding a domain component to an OID index is unnecessary.

Domain Procedure Library (mgdompl.p)

The Domain project created a library of common routines that interact with the domain table (dom_mstr). These routines must be used by new functionality needing to interact with domains. Below is a list of all these routines, including a brief description of their purpose, and examples of how to call them. If a project discovers that none of these routines exactly meet their requirements, then a new routine should be created in this library, so that it’s available to all other projects.

**ppDomainValidate** (input pDomain, output pError)
Validates that a domain is: not blank, not the SYSTEM domain, exists, and is active.

**ppDomainConnect** (input pDomain, output pError, output pParameter)
Calls ppDomainValidate, and also validates that the database associated with the Domain is connected.

**ppUserDomainValidate** (input pDomain, input pUserID, output pError)
Calls ppDomainValidate, and also validates whether a particular user has access to a particular domain

**ppUserDomDbValidate** (input pDomain, pUserID, output pError).
Calls ppUserDomainValidate, and also validates that the Domain belongs to the current database.

**ppDomHeader** (input pDomain, input pReturnLong, output pDomHeader)
Returns either the Domain name or Domain short name, depending on the setting of pReturnLong.

Pan-domain functionality

While most functionality runs completely within a single domain, this is not universally true. Functions related to non-domained tables (e.g. Printer Maintenance or Currency Maintenance) obviously apply to all domains. There is
also functionality relating to domained tables that operates across domains. This functionality is referred to as pan domain.

For example, in Batch Request Processor, the batch details are held at the domain level but you can process batches for any domain, as there is a domain selection field, allowing batch processing to be managed centrally.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Batch ID Domain</th>
<th>Batch ID Domain</th>
<th>Batch ID Domain</th>
<th>Batch ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>demo1</td>
<td>demo1</td>
<td>demo1</td>
<td>demo1</td>
<td></td>
</tr>
<tr>
<td>demo1</td>
<td>demo1</td>
<td>demo1</td>
<td>demo1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Repeat Processing: No</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pause Seconds Before Repeat: 300</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Another example is the new GL Consolidation process.

Moving forward, the high-level strategy is to put the majority of new pan domain functionality into the Distributed Order Management product. Therefore, one of the early requirement decisions for each project should be whether or not the new functionality should be designed to operate solely within a domain, or if it should work across multiple domains. If a project team desires to add pan domain functionality, the Product Management and Architecture teams must be involved early to insure that this functionality fits within the overall long term product strategy.

**Working with Site records**

The field si_db in the Site Master table (si_mstr) has been modified in the eB2.1 release to hold a domain value, as opposed to a database value. The meaning of the field remains the same – it tells MFG/PRO the true ‘owner’ of this site. In eB2 and earlier, this designated the remote database in which this site resided, and in which all inventory for that site was recorded. Now, it designates the domain in which this site resides (and therefore, the domain in which the inventory is recorded). Therefore, within a single domain, there will be ‘primary’ site records (sites that belong to that domain; si_db = si_domain; inventory for the site is in that domain), and there may be ‘connection’ site records (sites that belong to other domains; si_db <> si_domain; inventory for the site is in a different domain). Remember that si_domain identifies the actual location of the si_mstr record, while si_db identifies the true ‘owner’ of the site.

Careful consideration must be given when validating a site field. If a site must exist in the current domain (global_domain) to be valid (in other words it must be a ‘primary’ site), then the site must exist AND si_db must be equal to global_domain. In other cases (for example in DRP) it’s perfectly acceptable for a site to belong to a different domain.

It’s important to realize that this is functionally equivalent to how multi-database functionality worked in versions eB2 and earlier. If a site belonged to a different database, then the si_db designated which database it belonged to, and was considered a ‘connection’ site record (i.e. all inventory for that site resided in a different database). The Domain project made no changes other than to assign a Site to a Domain (instead of database), and assign a Domain to a database. Therefore, multi-database functionality is still supported. The logic just includes an additional check to see if the domain is in another database.

**Updating pgmi_mstr records for cross-domain functions**

For most programs the individual domain name can be displayed as part of the title bar. What is displayed depends on the setting of the Header Display Mode flag on the Security Control File (usrc_ctrl):

0 = program name and date,  
1 = program name and userID  
2 = domain short name and date  
3 = domain short name and userID

If options 2 or 3 are used and the program is a cross-domain function then the description “All Domains” should be displayed instead of the domain short name.
This is enabled by setting the multi domain flag (pgmi__qadl01) to “Yes” in 36.3.21.1 Program Information Maintenance in devmstr.

### Other Considerations

#### Is the Domain Active?

A domain cannot be deleted, but is instead made inactive. Any new designs should bear this in mind and introduce a check on the active flag (dom_active should be yes), where appropriate, to ensure no processing of any kind is done for domains which are inactive.

**Global variable global_db has been re-tasked**

For the domain design, the MFG/PRO global variable global_db was re-tasked, in order to reduce the coding effort required to complete the project. Prior to eB2.1, this variable was used to hold the current database that the QADDB alias was pointing to. However, for the domain functionality to work correctly, this variable has been re-tasked so that it holds the current domain value. So in the current version of eB2.1, both global_domain AND global_db hold the current domain value. While the goal is to eventually restore global_db to holding the database value, until that work is completed, all designs should NOT rely on the value in global_db. If the current domain needs to be referenced, then global_domain should be used. The new procedure gpmdas.p handles all domain and database switching.

Similarly, other fields that previously held a database value in versions up to eB2 now hold a domain value. These include si_db, pod_po_db, and so_conrep. As with global_db, it is intended to rationalize these field names in eB3, so that domain values are stored in ‘domain’ fields, and database values are stored in ‘_db’ fields. In most cases, this will simply mean renaming the current ‘_db’ fields to indicate that they hold domain values. If there’s a need to also store a database value, then a field with ‘db’ in the name will be added.

#### Examples

N/A

#### See Also

- **Architecture Design Model (QDN Download - QAD Internal Only)**
- **Design Specifications for Domain (QDN Download - QAD Internal Only)**
- **STD-0320 Domain Should Be Considered In All Queries (eB2.1 and up)**
- **STD-0323 Switching domains in procedure code**
STD-0004 SETUSERID use for Oracle

Summary
The SETUSERID function is un-supported and cannot be used to change the user ID and password of an Oracle login.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0004</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Data Handling Logic</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
The SETUSERID function is un-supported and cannot be used to change the user ID and password of an Oracle login.

Rationale
You cannot use the SETUSERID function to change the user ID and password of an Oracle login. (Progress Dataserver for Oracle Guide Version 9)
STD-0006 Allowed number of fields per table

Summary

A maximum of 1000 columns per table are allowed in QAD SE (MFG/PRO eB2.1 and above), QAD EE and other QAD products which use Oracle 9i, 10g or 11g. The limit may be slightly less depending on the Progress Oracle DataServer rules outlined below.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0006</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Progress, Oracle and Microsoft SQL Server allow different numbers of columns per table, but Oracle is the most restrictive. Therefore, the actual limit is determined by the Oracle database used on the particular version of MFG/PRO or the QAD product being developed. Beginning with Oracle 9i and continuing with versions 10g and 11g, the maximum number of columns allowed per Oracle table is 1000 (older Oracle versions had a lower limit). When designing a new database table or changes to an existing one, you must also consider the way Progress database features are mapped to equivalent Oracle features:

1. Each extent of a PROGRESS database array field equates to one column in the Oracle database**
   For example, a 100 element Progress array field is turned into 100 individual columns on Oracle
   ** See STD-0333 for further information and limitations about using Progress arrays in QAD products.
2. RECID numbers are stored in an extra column named PROGRESS_RECID in Oracle.

New QAD product releases therefore require the maximum number of fields in a Progress table to be something short of 1000 depending on the mapping rules above. For additional information, refer to the Oracle Database Reference manual, Database Limits topic for the Oracle version being used.

Rationale

One important distinction between Progress, Oracle and SQL Server is in the maximum column limit. Progress allows up to 32,000 fields to be defined per database file (table) and SQL Server permits 1024 columns per table, while Oracle allows up to 1000. Therefore, in order to design QAD product database tables to be compatible with Oracle and deployable on any of the three database types, the lower of these limits (1000 columns) must be met.

Examples

- A 10 column Progress table without any array fields requires 11 columns on Oracle (10 table columns + PROGRESS_RECID column).
- A 10 column Progress table containing one 100 element array requires 110 columns on Oracle (9 table columns + 100 columns for each array element + PROGRESS_RECID).
- A 1000 column Progress table without any array fields cannot be created on Oracle because there is no room to add the PROGRESS_RECID column.

See Also

STD-0050 Required Table Fields
STD-0051 Providing for Extra Customer Fields
STD-0307 Providing for Extra QAD Fields
STD-0333 Database fields with Extents are not supported
STD-0350 Maximum Record (Row) Size Limit
STD-0007 UNKNOWN values used in Unique Keys in Indexes Must Be Unique

Summary

Duplicate entries in a unique index, using the UNKNOWN value, are not allowed.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0007</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Table Access &amp; Storage, Field Access &amp; Storage, Logic for Dates, DB Indexes</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

In a Progress database it is possible to create duplicate entries in a unique index. This is possible when one or more fields in the entries are set to the UNKNOWN value. In an Oracle or SQL Server database however, it is not possible to create such duplicate UNKNOWN / NULL entries. An attempt to add a duplicate UNKNOWN / NULL key to an Oracle or SQL Server unique index results in a "duplicate unique key" error condition.

Applications where this situation is allowed to occur must be redesigned. To redesign your application, consider the following:

- To ensure that all entries are unique, perform a FIND or CAN-FIND to see if the record exists before creating it.
- If the index is not a primary index, and there is no other reason for it to be unique, change it to non-unique.
- If the UNKNOWN value is used to explicitly create duplicate entries in a unique index, add additional fields to the index schema definition to permit uniqueness and remove the use of the UNKNOWN value.

Be aware that date fields usually have their schema initial values set to the UNKNOWN value. A date field with an UNKNOWN value displays as <blank> on the screen and in reports. The resulting index/record creation errors often catch developers by surprise. In order to maintain uniqueness, you may need to do an explicit ASSIGN of a date field to ensure that the field is not set to the UNKNOWN value. For example, with the statement:

```
assign cal_date.
```

It's possible that a user might leave cal_date blank when prompted for input, in which case it gets assigned the UNKNOWN value. To prevent this, an explicit assign can be performed:

```
if input cal_date = ? then assign cal_date = today.
else assign cal_date.
```

Be aware that in the MFG/PRO schema there are some non-date fields that comprise unique indexes and that have an initial value of UNKNOWN. For example:

<table>
<thead>
<tr>
<th>Field-Name</th>
<th>Data-Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>atp_id_num</td>
<td>integer</td>
</tr>
<tr>
<td>gltr_rflag</td>
<td>logical</td>
</tr>
<tr>
<td>glt_rflag</td>
<td>logical</td>
</tr>
<tr>
<td>mps_record</td>
<td>integer</td>
</tr>
<tr>
<td>opgl_sequence</td>
<td>integer</td>
</tr>
<tr>
<td>rps_record</td>
<td>integer</td>
</tr>
<tr>
<td>trgl_sequence</td>
<td>integer</td>
</tr>
</tbody>
</table>
Rationale

Database constraints from Oracle and SQL Server impact the use of some of the features of Progress. It is necessary to take a least common denominator approach. Therefore, it is necessary to adhere to the Oracle and SQL Server limitation. The index of the MFG/PRO table “cal_det” is used for the following examples:

<table>
<thead>
<tr>
<th>Index-Name</th>
<th>Unique Field-Name</th>
<th>Format</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>cal_det</td>
<td>yes</td>
<td>cal_domain</td>
<td>x(8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cal_site</td>
<td>x(8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cal_wkctr</td>
<td>x(8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cal_mch</td>
<td>x(8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cal_ref</td>
<td>x(8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cal_start</td>
<td>99/99/99</td>
</tr>
</tbody>
</table>

Examples

Right

A record is created in the cal_det table. One of the index fields in one of the records is set to the UNKNOWN value. Since the new index entry will have an UNKNOWN value, it’s necessary to first see if the record already exists, and if so, handle appropriately.

```plaintext
if can-find(cal_domain = "dom1" and cal_det where cal_site = "10000" and cal_wkctr = "WELDING" and cal_mch = "ROBOT1" and cal_ref = "" and cal_start = ?)
then do:
  message "Sorry, record already exists".
  undo, retry.
end.
else do:
  create cal_det.
  assign cal_domain = "dom1"
  cal_site = "10000"
  cal_wkctr = "WELDING"
  cal_mch = "ROBOT1"
  cal_ref = ""
  cal_start = ?.
  message "Record added".
end.
```

Wrong

A record is created in the cal_det table. The field cal_start is assigned to the input screen value, which could be the UNKNOWN value. No determination is made whether the index entry already exists.

```plaintext
create cal_det.
assign cal_domain = "dom1"
  cal_site = "10000"
  cal_wkctr = "WELDING"
  cal_mch = "ROBOT1"
  cal_ref = ""
  cal_start.
```

See Also
STD-0044 Definition And Acceptable Values For Logical Variables In Progress
STD-0061 Make all fields 'Mandatory'
STD-0180 Every table must have a unique index & a primary index (pre-eB3)
STD-0327 Every table must have a unique index & a primary index (eB3 and up)
STD-0008 Indexes and Performance

Summary

Sometimes the Oracle Database Server does not choose the correct index to access data. There are steps that can be taken to correct this behavior.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0008</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress,Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Table Access &amp; Storage,Field Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Sometimes performance problems are caused by the Oracle database server selecting the wrong index to access the data requested by a SELECT statement. This can happen when:

1.1 The conditional statements are not in the order of the index defined,

1.2 The USE-INDEX phrase is being ignored because there is a JOIN being performed against two tables,

1.3 The wrong index is being selected due to the lack of index information,

**Determine Index Usage**

To find out what index is being used when one is not specified, compile the program and create a cross-reference list via the COMPILE statement's XREF option. In the cross-reference list, index usage is indicated by “SEARCH” reference types.

Solution: (in order of effectiveness)

2.1 Structure a WHERE clause to use the desired index's leading fields and have the field conditionals in the order of the fields in the index that should be used in the query.

2.2 In the case of a multiple table join query where the wrong index is being used, it is necessary to:

a) structure the WHERE clause as described in paragraph 2.1 above, or
b) add the USE-INDEX clause to the join query (contrary to popular belief, USE-INDEX clauses are not ignored inside a join), or

2.3 In the case of a single table query where the wrong index is being used, only add a USE-INDEX clause as a last resort.

**USE-INDEX Warning**

USE-INDEX is not recommended as it forces the use of a specific index which may not be appropriate or optimal for the query. It is better to write appropriate WHERE / OF / BY clauses to allow the native RDBMS optimizer to choose the best index.

2.4 For MFG/PRO versions eB2.1 and above, use the Oracle COST-based optimizer mode.

Additional Information:

The following Word attachment has further information about indexes and performance:

Indexes and Performance.doc
Examples

Right

Example 1

The WHERE clause conditions match the fields and component order of the desired pt_prod_part index, as does the BREAK BY clause, so USE-INDEX not required:

```sql
FOR EACH pt_mstr
  WHERE (pt_domain = global_domain)
  AND (pt_prod_line >= line AND pt_prod_line <= line2)
  AND (pt_part >= part AND pt_part <= part1)
  AND (pt_part_type >= type AND pt_part_type <= type2)
  AND (pt_group >= ptgroup AND pt_group <= group1)
  NO-LOCK
  BREAK BY pt_prod_line BY pt_part:
```

Example 2

JOIN in which The first query uses the correct index based upon the WHERE clause so no USE-INDEX option is required. However, the second join query's WHERE clause directs the optimizer to a different index (ssd_rec_site instead of ssd_det), so the WHERE clause needs an additional condition on the ssd_rec_site index component to direct the optimizer to select the ssd_det index or USE-INDEX can be specified, as shown here:

```sql
FOR EACH ptp_det NO-LOCK
  WHERE ptp_domain = global_domain
  AND ptp_part = part
  AND ptp_site <> site
  AND ptp_network > ""
  AND ptp_pm_code = "D":

  FOR EACH ssd_det NO-LOCK
    WHERE ssd_domain = ptp_domain
    AND ssd_network = ptp_network
    AND ssd_rec_site = ptp_site
    AND ssd_src_site = site
    AND (ssd_start >= today OR ssd_start = ?)
    AND (ssd_end >= today OR ssd_end = ?)
    USE-INDEX ssd_det:
```

Example 3

Same as Example 2 above, except a QUERY-TUNING clause is used instead of USE-INDEX:

```sql
FOR EACH ptp_det NO-LOCK
  WHERE ptp_domain = global_domain
  AND ptp_part = part
  AND ptp_site <> site
  AND ptp_network > ""
  AND ptp_pm_code = "D",
  EACH ssd_det NO-LOCK
    WHERE ssd_domain = ptp_domain
    AND ssd_network = ptp_network
    AND ssd_rec_site = ptp_site
    AND ssd_src_site = site
    AND (ssd_start >= today OR ssd_start = ?)
    AND (ssd_end >= today OR ssd_end = ?)
    query-tuning(NO-INDEX-HINT
      HINT "INDEX_ASC(T0 SSD_DET##SSD_DET)"
    ):
```

Example 4

Single table query uses USE-INDEX because its WHERE clause would otherwise direct the optimizer to a different
index (tr_addr_eff instead of tr_nbr_eff):

```sql
FOR EACH tr_hist
  WHERE tr_domain = global_domain
  AND tr_nbr = nbr
  AND tr_line = line
  AND tr_type = "ORD-SO"
  AND tr_addr = cust
USE-INDEX tr_nbr_eff
EXCLUSIVE-LOCK:
```

**Example 5**

Configure Oracle initialization parameter (in init.ora or SP file) to use the cost-based optimizer (for eB2.1 and above):

```sql
optimizer_mode=ALL_ROWS
```

**Wrong**

**Example 1**

The order of conditions in the WHERE clause does not match the index specified or the BREAK BY clause:

```sql
FOR EACH pt_mstr
  WHERE (pt_domain = global_domain)
    AND (pt_part >= part AND pt_part <= part1)
    AND (pt_prod_line >= line AND pt_prod_line <= line2)
    AND (pt_part_type >= type AND pt_part_type <= type2)
    AND (pt_group >= ptgroup AND pt_group <= group1)
NO-LOCK USE-INDEX pt_prod_part
BREAK BY pt_prod_line BY pt_part:
```

**Example 2**

The second join query’s WHERE clause directs the optimizer to a suboptimal index (ssd_rec_site instead of ssd_det):

```sql
FOR EACH ptp_det NO-LOCK
  WHERE ptp_domain = global_domain
    AND ptp_part = part
    AND ptp_site <> site
    AND ptp_network > ""
    AND ptp_pm_code = "D",
EACH ssd_det NO-LOCK
  WHERE ssd_domain = ptp_domain
    AND ssd_network = ptp_network
    AND ssd_rec_site = ptp_site
    AND ssd_src_site = site
    AND (ssd_start >= today OR ssd_start = ?)
    AND (ssd_end >= today OR ssd_end = ?):
```

**Example 3**

As in Example 2 above, the wrong index is being used here (WHERE clause leads to tr_addr_eff instead of tr_nbr_eff):
FOR EACH tr_hist
  WHERE tr_domain = global_domain
  AND tr_nbr = nbr
  AND tr_line = line
  AND tr_type = "ORD-SO"
  AND tr_addr = cust
EXCLUSIVE-LOCK:

See Also

N/A
STD-0011 Maintaining Sequences

Summary

When creating a sequence you must create a language detail for the sequence description and update Database Sequence Initialization (utsequp.p) program.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0011</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Translatable Strings, DB Sequences</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

You can define sequences in the ORACLE or SQL Server database and access them through the PROGRESS DataServer. Drop sequences that are no longer used.

Exceptions

All sequences should be initialized in utsequp.p except:

- uusg_sq01 used by the User Count Audit (UCA) module for QAD products in the Master Bundle (project 10479).

Description

Creating a Sequence:

You must create a language detail through devmstr for each sequence created. Set the fields in the lngd_det record as follows:

<table>
<thead>
<tr>
<th>Devmstr Label</th>
<th>lngd_det field</th>
<th>lngd_det field value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Set</td>
<td>lngd_dataset</td>
<td>&quot;sch_maint&quot;</td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>lngd_field</td>
<td>sequence name (e.g. tr_sq01)</td>
<td></td>
</tr>
<tr>
<td>Numeric Code</td>
<td>lngd_key1</td>
<td>&quot;0&quot;</td>
<td></td>
</tr>
<tr>
<td>Mnemonic</td>
<td>lngd_key2</td>
<td>&quot;S&quot;</td>
<td></td>
</tr>
</tbody>
</table>

When the sequence is used on only one table against one field name of the field which will use the value generated by the sequence in <table name>,<field name> format (e.g. tr_hist.tr_trnbr) when the sequence is used on many tables and updates related fields across multiple tables enter a meaningful description of the field type being updated, for example 'keyid' or 'Session ID'.

<table>
<thead>
<tr>
<th>Programmer Comments</th>
<th>lngd_desc</th>
<th>Enter a short description, and include that the Label (lngd_translation) is not to be translated.</th>
</tr>
</thead>
</table>
You must also update the 36.16.17 Database Sequence Initialization (utsequp.p) program. This routine will update the sequence values of a database based on the record content. This routine must be updated every time a new sequence is added to the database or if the usage of that sequence has changed. It is used to synchronize the sequence values with the data stored in the database.

**Deleting a Sequence**

1. When a table is dropped that uses a sequence to initialize one of its fields, then that sequence must also be dropped.
2. A sequence should only be dropped when it is used only by a single table that is being dropped. A sequence must not be removed when it is used for multiple tables and one or more of those tables still remain.
3. When a sequence is deleted, the corresponding code for that sequence in utsequp.p must also be removed.

**Accessing a sequence in ORACLE**

1. You must define sequences in ORACLE if you've defined them in PROGRESS. Sequences are database objects and so are built in Oracle along with other database objects (tables, indexes) necessary for the application. The PROGRESS DataServer uses the sequences and other objects on ORACLE.
2. The Progress 4GL functions NEXT-VALUE and CURRENT-VALUE work the same for ORACLE-defined sequences as PROGRESS-defined sequences, with the following limitations:
   - NEXT-VALUE must be executed before CURRENT-VALUE can be used in the same session.
   - This means that sequences on Oracle are incremented before use, while on Progress CURRENT-VALUE obtains a sequence's value without incrementing it. This is another of the "Oracle as lowest common denominator" differences to which QAD products must adhere, so logic must be written to expect sequences will be incremented before they are used.

   - You cannot reset a sequence value with CURRENT-VALUE. Changing a sequence's value on Oracle cannot be done programmatically. This limitation affects programs that change database sequence values, such as installation programs, conversion programs, and 36.16.13 Sequence Maintenance (mgsqmt01.p). To resolve the limitation, the sequence must be physically dropped then re-created with a new initial value on ORACLE. Another solution is to create a program loop that increments the sequence value by 1, provided the sequence is known to be 0.

**Accessing a sequence in SQL Server**

1. Microsoft SQL Server does not use sequences. The Progress DataServer for MSS emulates Progress sequences on SQL Server by creating MSS stored procedures that mimic a sequence's function.
2. The functions NEXT-VALUE and CURRENT-VALUE work the same on SQL Server databases as with PROGRESS-defined sequences, with the following limitations:
   - Unlike Oracle, the SQL Server DataServer allows the use of CURRENT-VALUE before NEXT-VALUE within a session.

   However, since the Oracle limitation is more restrictive, applications must be written to expect sequences will be incremented before they are used (NEXT-VALUE must always be executed before CURRENT-VALUE). This is important in order to maintain database independence of QAD code.

**Rationale**

Keep the Database Sequence programs up to date. Provide a mechanism for initializing sequences. Oracle DataServer limits the use of the CURRENT-VALUE function to update sequences.

**Examples**

Create a language detail for the new sequence with the following field values when the sequence updates only one table:
Create a language detail for the new sequence with the following field values when the sequence updates many tables:

```plaintext
lngd_dataset = "sch_maint"
lngd_field = "gp_sq01"
lngd_key1 = "0"
lngd_key2 = "S"
lngd_translation = "keyid"
lngd_desc = Translation not required. This is used for tables with a field keyid.
```

Update the program utsequp.p with the following logic when the sequence updates only one table:

```plaintext
/*
 * SEQUENCE: tr_sq01
 */
lcl_trnbr = 0.

for last tr_hist where tr_trnbr >= 0 no-lock:
  lcl_trnbr = tr_trnbr.
end.

run p-update-sequence
(input "tr_sq01").
```

Update the program utsequp.p with the following logic when the sequence updates many tables:

```plaintext
/*
 * SEQUENCE: gp_sq01
 */
lcl_trnbr = 0.

/*
 * THE SEQUENCE GP_SQ01 IS USED TO ESTABLISH THE KEYID FOR MANY TABLES
 */

/* THE FOLLOWING TABLES BELONG TO THE KANBAN SYSTEM. */
for last knbadc_hist no-lock where knbadc_keyid >= 0:
  lcl_trnbr = knbadc_keyid.
end.
for last knbd_det no-lock where knbd_keyid >= 0:
  lcl_trnbr = max(lcl_trnbr , knbd_keyid).
end.
for last knbfd_det no-lock where knbfd_keyid >= 0:
  lcl_trnbr = max(lcl_trnbr , knbfd_keyid).
end.
for last knbism_det no-lock where knbism_keyid >= 0:
  lcl_trnbr = max(lcl_trnbr , knbism_keyid).
end.

run p-update-sequence
(input "gp_sq01").
```

See Also

STD-0183 Sequence Naming
STD-0012 Always Trim Substrings

Summary

Use TRIM function when using the SUBSTRING function.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Table Access &amp; Storage, Data Handling Logic</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

The output of the SUBSTRING function should always be trimmed using the TRIM function.

Rationale

Substrings are the cause of many "record not found" errors. If a value does not meet the length passed to the SUBSTRING function, ORACLE fills it with blanks. If this blank-padded value is compared to another value that is not blank-padded, the comparison fails.

Examples

Right

```sql
for first pt_mstr where pt_part = trim(substring(part, 10, 8)) no-lock:
end.
```

Wrong

```sql
for first pt_mstr where pt_part = substring(part, 10, 8) no-lock:
end.
```
STD-0013 Table Name and Description

Summary

Table names are a maximum of 14 characters long and must have a table description.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0013</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Tables</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Known Exceptions:

Currently, many (118) table names in the QAD Financials schema exceed the 14 character limit. No new tables with names exceeding 14 characters should be added to the Financials database.

Description

Table Names (All products):

1. Table names are a maximum of 14 characters long and consist of a prefix, an underscore, and a suffix.
2. Table names should be entered in lower-case only.
3. "Dump file" name rules:
   - Products running Progress OE10.1 or earlier versions:
     - The table name must be unique within the first 8 characters for purposes of dumps and loads.
     - The first 8 characters should form the "dump file" name, which must be correctly entered for the table when performing data dictionary maintenance. Progress limits the dump file name to 8 characters in data dictionary maintenance.
   - Products running Progress OE10.2 or later versions:
     - The entire table name should form the "dump file" name, which must be correctly entered for the table when performing data dictionary maintenance. Progress 10.2 now allows the entire table name as the dump name in data dictionary maintenance.
     - The "dump file" name must not be changed once it has been released as it makes it more difficult to use dump files from previous versions of the schema. This includes tables which were originally created on older Progress versions where the dump file name limit was 8 characters but which now run on Progress 10.2 or newer; do not change existing dump file names to accommodate the relaxed OE10.2 length limit.
4. Do not begin any table name with the letter "z". This is reserved for user tables.
5. The prefix is a minimum of 2 and a maximum of 8 characters long.
6. The prefix must be unique to each table for purposes of column naming.
7. The suffix is a minimum of 2 characters long and historically has described the type of table. Some common conventions are "mstr" for master & header tables, "det" for detail tables below header tables, "ctrl" for control tables and "hist" for history tables. The designer can use any suffix, but should try to use the common ones if they fit the table being created.
8. Tables names not only must be unique within the current set of tables in any given product release, but they also must be unique across all currently supported and past releases of the product. This is necessary because data problems can arise when a table from a previous release is dropped, and then a new table is added with the same name. Customers upgrading from the previous release may not have accounted for the difference in data.
   - For MFG/PRO schema, designers can check the uniqueness of a new table name by searching for an occurrence of the name in the mfgempty.df, admempty.df and hlpempty.df files in the current release's $STGOBJ directory, and by checking for the name in the list of dropped tables found in $CUSTOM/drop.lst and obsoleted/renamed tables in /qad/dbs/localdbs/convdbs/src/rename.rf

Table Names (QAD Financials schema):
1. Table names are a maximum of 14 characters long.

2. Table names will be entered in mixed case format with a capital letter indicating the start of a new word.

3. Tables that are grouped in a single class have a dependency in their naming: the child-tables should be named like this:
   `<parent-table-name><suffix-that-identifies-the-child>`
   Do note that this rule is also applicable in case you have a child that is the parent to another table in the class.

4. The "dump file" name should consist of two or more letters from each 'word' (lower case) and, as of Progress OE10.2, may now exceed 8 characters.
   - Example: Table Name: WorkObjScope  Dump Name: woobso
   - Example: Table Name: AddressType  Dump Name: adtype
   - The "dump file" name must not be changed once it has been released as it makes it more difficult to use dump files from previous versions of the schema. This includes tables which were originally created on older Progress versions where the dump file name limit was 8 characters but which now run on Progress 10.2 or newer; do not change existing dump file names to accommodate the relaxed OE10.2 length limit.

5. Do not begin any table name with the letter "z". This is reserved for user tables.

6. Tables names not only must be unique within the current set of tables in any given Financials release, but they also must be unique across all Financials releases, including past releases. This is necessary because data problems can arise when a table from a previous release is dropped, and then a new table is added with the same name. Customers upgrading from the previous release may not have accounted for the difference in data.

Table Description:

1. The description should be in noun form, rather than a verb or sentence.
2. The description should be short, typically fewer than 5 words long.
3. The description should not use the word "file" to describe a database table.
4. The description should avoid the use of hyphens to separate parts of the description. Hyphenated words such as Cross-Reference are permitted.
5. Do not put "NOT USED" or other such text in the description or label. It will be needlessly translated.
6. Avoid the use of abbreviations and acronyms in the table description.

The label attribute on each table can be filled in, but it is not necessary since Knowledge Engineering does not use that field at this time.

Rationale

Oracle 10g and 11g allow table and index names up to 30 bytes long. However, the real limit on table name size comes from the Progress Oracle DataServer conversion utility which adds a column named PROGRESS_RECID to the Oracle table then creates an index named `<tablename>##progress_recid`. Since Oracle indexes are also limited to 30 characters and the "##progress_recid" string consumes 16, the table name used in the conversion cannot exceed 14 characters. Keep in mind that when the Progress Oracle DataServer conversion utility creates an index in Oracle, it will follow the naming convention `<tablename>##<indexname>`. If the table name and the index name are both 14 characters, the total length will be 30. A 14 character table name limit allows us to continue to use Progress DataServer tools to produce working and standardized schema for Oracle & SQL Server automatically with little need for manual intervention.

![Warning: Progress DataServer tools will truncate Oracle and SQL Server object names exceeding 30 characters in length, resulting in different schema object names between Progress, Oracle & SQL Server.]

Table descriptions are an expanded form of the table name that are extracted from the schema definitions and used in various Knowledge Engineering publications. The description is published exactly as it is entered in the database schema; spelling, grammar, and punctuation count.

Examples

Right

Table Names:
- bl_ctrl
- pcd_det
ma_mstr
ActivityCode (QAD Financials)
PreInvoiceBank (QAD Financials)

**Table Description:**
Master Bill of Lading Control
Purchase Order Line Cost Element Detail
Mirror Account Master
Activity Code
Pre-Invoice Bank

**Wrong**

**Table Names:**
zbl_ctrl
pcdet
ma_master_control
PreInvoiceLineDet
SOCONTROL

**Table Description:**
Contains the next Master Bill of Lading number
PO Line Cost Element Detail
General Ledger – Mirror Account Master

**See Also**

STD-0014 Field Names
STD-0286 Index Names
STD-0183 Sequence Naming
STD-0014 Field Names

Summary

Field names are limited to 22 characters and must be named and labeled appropriately.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0014</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>5</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields, Translatable Strings</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

All QAD Products:

1. Field names are limited to a maximum of 30 characters.
2. Don’t give similar field names to fields that are not similar.
3. Use labels consistently. If two fields are similar, (e.g. jcd_surf_spot, wug_surf_spot), then the labels should be the same, “Favorite Surf Spot”.
4. Don’t use the same label for dissimilar fields.
5. Leave column labels undefined (?) if they are the same as the side label.
6. Try to keep column labels to the format width by stacking or abbreviating the column label. This allows reports to contain as many columns as possible. However, when the format width is very short (i.e. less than 4), make the abbreviation sensible. Nonsensical labels do no good since they must be overridden in every report, and consistency is lost.
7. Capitalize each word of a label. If abbreviations are necessary, use only approved abbreviations (refer to standard STD-0037 Abbreviations). There are some exceptions to this rule. For example, “Hours per Period” would be correct even though ‘per’ is not capitalized.
8. Only use “Effective Date” as a label for financial fields. “Effective Date” is translated into “Posted Date” for Europeans and is very confusing if that is not the real meaning of the translated date label. Do not use “Effective Date” for non-Financial fields as that will confuse the issue when translating.
9. Only use “Item Number” as a label for Item fields. “Part” is not translated into “Part” for Europeans and is very confusing.

MFG/PRO Fields:

MFG/PRO Field names (qaddb, qadadm) are limited to a maximum of 30 characters** and must be in lower-case only.

**Existing Array fields are limited to 25 characters (Note: New array fields are prohibited. See STD-0333)

In addition to the rules in the “All QAD Products” section above, the following also apply:

1. Field names will begin with the table prefix (e.g. so_nbr in so_mstr) except for OID fields which always start with “oid_”. The main OID field must be named “oid_<table_name>” where <table_name> is the name of the Progress table. Foreign key OID fields have additional naming requirements. See STD-0325 Use OID fields as foreign keys to relate qaddb tables (eB3 and up) for more information.
2. Name your fields consistently across tables, particularly those that are used for joining tables. Please refer to the following standards for more information:
   - STD-0325 Use OID fields as foreign keys to relate qaddb tables (eB3 and up)
   - STD-0297 Reuse attributes of common schema fields
   - STD-0302 Define related fields with similar schema properties
3. Use label functionality in programs so field labels are retrieved from the label tables.

Financials Fields:

QAD Financials DB Field names are limited to a maximum of 30 characters and should be mixed-case only.

In addition to the rules in the “All QAD Products” section above, the following also apply:

1. Field names will be entered in mixed case with a capital letter indicating the beginning of a word
2. Do not use "_" (underscore) with the exception of ID fields.
3. Abbreviations may be used but should be avoided where possible
4. Every table requires a field named "<table-name>_ID". The primary index on this table will just hold this single field. For example: BusActivity_ID
5. ID fields required for foreign-key relationships:
   • If parent-table <A> has a relation to child-table <B>, then <B> will hold a field named "<A>_ID" that represents this relation. For example: Parent table Posting is related to child-table PostingLine, so PostingLine contains field Posting_ID as a link back to the Posting table.
   • If there are multiple such relations between these tables, then <B> will hold fields named "<Functional-differentiation-Identification><A>_ID" that reflect the different relations. For example, child-table CDocument contains field CreationUser_ID to link to parent User table's User_ID field.

6. If adding a new field that represents data 'owned' by MFG/PRO, the field name should begin with Mfg. For example: MfgSalesPersonCode
7. Fields of type 'logical' start with the table-name followed by "Is", followed by a word that reflects the function of the field. For example: UsrIsActive and WorkObjectIsSendMail

**AIM (QAD Warehousing) Fields:**

QAD Warehousing field names are limited to a maximum of 30 characters.

In addition to the rules in the "All QAD Products" section above, the following also apply:

1. Field names will be entered in mixed case with a capital letter indicating the beginning of a word.
2. Do not use "_" (underscore).
3. Abbreviations may be used but should be avoided where possible.

**Rationale**

The 30 character field name length limit comes from Oracle which (in versions 10g and 11g) allow column names up to 30 characters long. For existing Progress array fields, however, the real limit on field name size comes from the Progress Oracle DataServer conversion utility which adds several characters to the field name. The DataServer appends two pound signs "##" and a numeric identifier of up to three digits (##999) to Progress array fields whose extents must be converted to Oracle columns (for example: qad_charfld##1, msg_explanation##10, usrw_charfld##15, etc.). Subtracting this 5 character maximum array field suffix from the Oracle column name limit of 30 leaves 25 characters for array field names. All other data types can have 30 character field names.

Note: Progress versions prior to OE10 also added a 3 character U## prefix to case-insensitive character datatype fields used in indexes. This 3 character "shadow column" prefix, coupled with the 5 character array field suffix meant that MFG/PRO versions eB2 and earlier, which used Progress v9 and earlier, were limited to field names of just 22 characters. However, this U## shadow column restriction does not apply to MFG/PRO versions eB2.1/QAD SE and above (93/QAD EE) which use Progress OE10.
• QAD should enforce a field name length limit in its products so we have control over actual field names instead of allowing Progress to create field names as it truncates long (30+ character) names to fit Oracle.
• Allowing the Progress DataServer to truncate/change field names complicates and adds time to the development of QAD product conversion processes, especially the In-Place conversion process for Oracle databases. The risk of Progress changing its name truncation algorithm or output format over time, such as if the same input field name now produces a different output name than in the past, would lengthen the time required to create QAD conversion processes and programs during the code cutoff release phase.
• Oracle schema comparison tools cannot determine if a different field name is truly a new field added to a release or simply a truncated/changed name for a pre-existing field. The tools would report all such fields as new and produce accompanying UPDATE TABLE ... ADD FIELD commands in SQL scripts, along with DROP statements for the old field names. These scripts would then have to be manually altered to correct these commands. This would be a significant change to a development process which we’ve used successfully for many years and across many MFG/PRO service pack and version releases.
• Customers also use Oracle and other 3rd-party ODBC tools for customization and reporting. Having field names which differ between the Progress database (as well as our published Database Definitions manuals) and the names used on Oracle would make it difficult for customers to integrate their customizations or use other applications with QAD databases.

Examples

Right

MFG/PRO Database

| table: jcd_mstr | field: oid_jcd_mstr | Main OID field |
| field: jcd_surf_spot | (Favorite Surfing Spot) |
| table: wug_mstr | field: oid_wug_mstr | Main OID field |
| field: oid_jcd_mstr | Foreign key OID field |
| field: wug_surf_spot | (Favorite Surfing Spot) |

MFG/PRO Financials Database:

| table: Activity | field: ActivityDate |
| field: ActivityIsTimeRegistration |
| field: ActivityIsCosted |

Wrong

| table: jcd_mstr | field: jcd_oid_jcd_mstr | (shouldn't have prefix) |
| field: jcd_surf_spot | (Favorite surfing spot) |
| table: wug_mstr | field: wug_oid_wug_mstr | (shouldn't have prefix) |
| field: wug_oid_jcd_mstr | (shouldn't have prefix) |
| field: wug_srf_spot or wug_surf_spt, etc. | (doesn't match field name in jcd_mstr) |

MFG/PRO Financials Database:

| PaymentConditionStagedPercentage | (too many characters) |

See Also
STD-0351 Supported Progress Data Types
STD-0030 Label Names
STD-0037 Abbreviations for Field Labels
STD-0297 Reuse attributes of common schema fields
STD-0302 Define related fields with similar schema properties
STD-0286 Index Names
STD-0013 Table Name and Description
STD-0325 Use OID fields as foreign keys to relate qaddb tables (eB3 and up)
DST-0005 Developing with OID fields (eB3 and up)
STD-0017 Using Shared frames, streams etc

Summary

Use an include file to define the shared Progress elements. 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.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0017</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Procedure Structure, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All new development activities</td>
</tr>
</tbody>
</table>

⚠️ This standard is not applicable to shared temp-tables or variables because shared temp-table and variables should be avoided by passing them as parameters to other programs or procedures.

Description

If shared Progress elements like 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 can not be sent in as input parameter to another programs/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 which can not be sent as parameters.

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 etc. See the development standard on 'Source File Formatting - Header' for more information.

Rationale

By keeping shared elements in one physical file, code maintenance can be simplified.

Examples

Right
Include file: 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)"

program2.p
    /* program2.p - Subprogram */

    { incfile.i "shared"

Wrong

program1.p
    define new shared frame a.
    define new shared stream rpt.

    form
        field1
        field2
        with frame a.

        "<more code here>

        {gprun.i "" program2.p"" "(output error_flag)"

program2.p
    /* program2.p - Subprogram */

    define shared frame a.
    define shared stream rpt.

    form
        field1
        field2
        with frame a.

        /* program logic */

See Also

N/A
STD-0018 Use `getFrameTitle()` function to assign Frame Titles (eB and up)

Summary

Assign frame titles by using `getFrameTitle()` function.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0018</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Translatable Strings</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities (MFG/PRO versions eB and Up)</td>
</tr>
</tbody>
</table>

Description

`getFrameTitle()` function returns a frame title for a specified term from Label Master table. A frame title is a string with a single white space in both the string's lead and trail positions.

Label Master Term should be passed as a parameter to this function and must be supported by Label Master records.

Rationale

Frame Titles will not be translated into foreign language if `getFrameTitle()` function is not used.

Examples

Right

```plaintext
form
   cm_taxable colon 35
   cm_taxc no-label
   cm_pr_list2 colon 35
   cm_pr_list colon 35
   cm_fix_pr colon 35
   cm_class colon 35
   cm_partial colon 35
with frame b2 title color normal {getFrameTitle("CUSTOMER_DATA",78)}
side-labels width 80 attr-space.
```

Wrong
/* ********** Begin Translatable Strings Definitions ********* */

&quot;SCOPE-DEFINE adcsmt02_i_4 " Customer Data "
/* MaxLen:78 Comment: Frame Title */

/* ********** Begin Translatable Strings Definitions ********* */

form
    cm_taxable colon 35
    cm_taxc no-label
    cm_pr_list2 colon 35
    cm_pr_list colon 35
    cm_fix_pr colon 35
    cm_class colon 35
    cm_partial colon 35
with frame b2 title color normal (&quot;adcsmt02_i_4)
side-labels width 80 attr-space.
STD-0019 Report Header Problem

Summary

There is a common coding problem with spacing between a header and its report which can be solved by a coding technique.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0019</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Reports/Inquiries</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Report header spacing is not always consistent. A common coding problem with spacing between a header and its report and a coding solution is described below:

1. Problem

Many of the reports found in MFG/PRO have been coded incorrectly. They use multiple frames to produce blank lines between sections of the report. The idea of using multiple frames is perfectly adequate, but the code usually reflects that the "view"ing of the frames is done inappropriately.

1.1 The first frame is viewed in the correct position: It is out of any major loop and it is defined as a form with a header attribute.

1.2 Every time a new page is being printed this frame will be displayed, along with the mphead.i header.

1.3 The second frame is usually "view"ed upon every iteration of the major loop. It is used to separate the different sections of the report. This is where the problem lies. There are times when the major loop will begin displaying data directly after the two headers. The second frame is a form with only a skip(1) statement. This frame would be displayed after the two headers, thus causing two blanks lines following the header, instead of having a single blank line.

2. Solution

Remove the "view"ing of the second frame from the top of every iteration of the major loop. Make the report generator smart about its location on the display, by checking the line-counter. If the line-counter is greater than the header length, then the report is beyond the headers and it is safe to "view" the frame.

Rationale

Current coding practice of report does not handle spacing between report header and detail correctly. This standard provides a solution to handle the problem consistently.

Examples

Right

Use the following line, when determining the time to "view" the second frame:

```plaintext
if line-counter > 4 then view frame skipline.
```

Here line-counter is compared against 4, which is the number of lines in the header. "View"ing only takes place, if a blank line is needed.
/* view frame phead2. */ /* Remove the 'view'ing of the second frame */
print_customer_header = yes.
if include_allocated and
include_picked and
include_shipped and
include_unprocessed
then do:
if line-counter > 4 then view frame skipline. {sosorp.i}
print_customer_header = no.
end.

Wrong

view frame phead2.
print_customer_header = yes.
if include_allocated and
include_picked and
include_shipped and
include_unprocessed
then do:

{sosorp.i}
print_customer_header = no.
end.
STD-0020 Using Language Detail

**Summary**

Use the Language Detail table and standard include procedures to manage the storage and retrieval of language-specific mnemonics.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0020</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Data Handling Logic, MFG/PRO Application, Translatable Strings, Character UI Apps, Devmstr</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

**Description**

Use language-specific mnemonics for both input and output using the Language Detail table and a set of routines provided with this standard. Use the maintenance routines to create and maintain the US English entries in Language Detail (lngd_det); use the interface routines to provide Language Detail services to your programs.

The functionality is implemented as follows

1. **For Input:**
   a. Prompt the users for native language mnemonic;
   b. Provide new lookup browse (instead of a scrolling window). (The routines swingd*.p and swlang*.p should no longer be used to create new scrolling windows for Language Detail.) Pass mnemonic to interface routine to obtain numeric code and label;
   c. Use the internal numeric code for processing and/or storage;
   d. Use the label for display.
2. **For Output:**
   a. Get internal numeric code from storage (internal or external).
   b. Obtain the translated text from Language Detail using interface routine.
   c. Output the translated text (mnemonic or label).
3. **Interface Routines**
   a. gplnga2n.i - Alpha To Numeric - Convert mnemonic to code
   b. gplngn2a.i - Numeric To Alpha - Convert code to text
   c. gplngv.i - Validation - Validate mnemonic
4. **Maintenance Routines**

   ! The database field names used in these routines may be confusing, as the database field names (when using Ctrl-F) do not correspond to the labels on the screen. For example, the field Numeric Code is the database field lngd_key1, the Mnemonic field is the database field lngd_key2 and the field Label is the database field lngd_translation. The Translatable Text field is not accessible. It is actually a section title indicating the fields below it are translatable.

   1. Development version: mgingdmt.p
      * Allows create, delete and update of Language Detail records.
   2. User version: mgingumt.p
      * Records are created in base language by default
      * Does not allow creation or deletion of Language Detail records.
      * Allows update of mnemonic and full translatable text only.

2. **Support Mode**

Support mode can be turned on using the "Lngd_det Support Mode" menu option. While in this mode, the Language Detail interface mechanism will perform as follows:

5.1 Routines that accept a mnemonic as a parameter will expect an internal numeric code instead.
5.2 Routines that return a mnemonic will return a numeric code in place of it.

The contents of Language Detail are NOT ignored while in this mode. Numeric codes are mapped to themselves using the same search method that
is used in normal mode.

**Rationale**

Support mode can be a useful troubleshooting feature. Constant character values hard-coded in the logic are not translated. Using language mnemonics allows language-specific data expected in the code to be presented to the user in their own language and remain language independent for interpretation within the code.

**Examples**

**Right**

This example illustrates how language-specific data can be stored in the database. When it is displayed to the user it will be translated into their language.

```plaintext
define variable attributelabel as character format "x(24)" no-undo.
define variable attributemnemonic as character
    label "Attribute" no-undo.
define variable validmnemonic as logical no-undo.

form
    . .
    attributemnemonic colon 20
    attributelabel
    . .
with frame a.

/* GET LANGUAGE DETAIL FOR ATTRIBUTE */
gplngn2a.i
    &file = "slr_mstr"
    &field = "slr_attribute"
    &code = slr_attribute
    &mnemonic = attributemnemonic
    &label = attributelabel
}
display
    . .
    attributemnemonic
    attributelabel
    . .
with frame a.
```

**Wrong**
set
  attributemnemonic
with frame a.

/* VALIDATE MNEMONIC */
{gplngv.i
  &file = "slr_mstr"
  &field = "slr_attribute"
  &mnemonic = attributemnemonic
  &isvalid = validmnemonic
}
if not validmnemonic then do:
  /* RAISE ERROR MESSAGE */
end.

/* GET LANGUAGE NEUTRAL CODE FOR MNEMONIC AND STORE IN TABLE */
{gplnga2n.i
  &file = "slr_mstr"
  &field = "slr_attribute"
  &code = slr_attribute
  &mnemonic = attributemnemonic
  &label = attributelabel
}

display
  attributelabel
with frame a.

This example illustrates hard-coded data stored in the database that is displayed back to the user. The data would not get translated for other languages and would always be displayed in English.

set
  slr_attribute
with frame a.

if slr_attribute = "1" then
  slrd_attrib_code = "PRODLINE"
else if slr_attribute = "2"
  slrd_attrib_code = "ITEMTYPE"
else if slr_attribute = "3"
  slrd_attrib_code = "PUR-MFG"
else
  slrd_attrib_code = "NONE"

display
  slr_attribute
  slrd_attrib_code
with frame a.
STD-0022 Database character set

Summary

Choose the correct Progress database code page for your QAD application. When the application's primary database is Oracle or SQL Server, select a compatible combination of Progress code page for the schema holder along with Oracle character set or SQL Server collation when setting up QAD products.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0022</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Environment</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Character processing is a critical function for exchanging information across language barriers. A character set is the set of all symbols used to represent a language in writing, and a collation is the order that a language's symbols are listed in when sorted sequentially. Code pages (also called encoding schemes) map character sets and collations to numeric codes a computer understands.

Database systems use character sets encoded into code pages to support internationalized or globalized (multilingual) applications by enabling you to store, process and retrieve data in native languages. Encoded character sets ensure that database utilities, error messages, sort order, and date, time, monetary, numeric and calendar conventions can adapt to any native language and locale. The result is an application that can be accessed and run from anywhere in the world simultaneously and render the user interface and process data in the native users' languages and locale preferences. You must specify an encoded character set when creating a database. Choosing a character set determines what languages can be represented in the database. Progress, Oracle and SQL Server support a large number of single-byte and multibyte encoding schemes based on national and international standards. Typically, encoded character sets support a fixed group of related languages. This limited group of supported languages is called restricted multilingual support. For example, the ISO 8859 character set series was created to support different European languages. More recently, universal character sets have emerged to enable greatly improved solutions for multilingual support. Unicode is one such universal character set. Unicode is an industry standard that enables text and symbols from all languages to be consistently represented and manipulated by computers in a single character set. Unicode provides a unique code value for every character in most of the spoken languages of the world and so provides unrestricted multilingual support.

Consider the following questions when choosing a database character set:

- What languages does the database need to support now?
- What languages does the database need to support in the future?
- Is the character set available on the database server's operating system?
- What character sets are used on clients?

Several character sets may meet your current language requirements. Consider future language requirements when choosing a database character set. If you expect to support additional languages in the future, then choose a character set that supports those languages to prevent the need to migrate to a different character set later. When you use a character set that supports a limited group of languages, your database has restricted multilingual support. If you need unrestricted multilingual support, then use a Unicode character set for the database.

Documentation References:

Obtain appropriate character set/codepage values and implementation instructions (startup parameters, environment variables, etc.) for each database type from the following books:

- Progress OpenEdge Development: Internationalizing Applications
- Progress OpenEdge Data Management: DataServer for Oracle
- Progress OpenEdge Data Management: DataServer for Microsoft SQL Server
- Oracle Database Globalization Support Guide
- Microsoft SQL Server Installation Guide
When using a Progress DataServer to access data stored in an Oracle or SQL Server database, make sure to select a code page for the Progress schema holder which is compatible with the Oracle database's character set or SQL Server database's collation table.

**Rationale**

Building databases with appropriate character sets and codepages helps ensure data corruption will not occur due to codepage conversions when language data is stored or retrieved by the application, maximizes system performance by minimizing character conversions between languages, and makes the application appear to users as though it were developed locally for their language and locale preferences.

**Examples**

**Progress:**

<table>
<thead>
<tr>
<th>Language/Region</th>
<th>PSC Code Page</th>
<th>PSC Collation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Western European (includes US English)</td>
<td>ISO8859-1</td>
<td>Basic</td>
</tr>
<tr>
<td>Eastern European</td>
<td>1250</td>
<td>Czech Hungarian Polish</td>
</tr>
<tr>
<td>Japanese</td>
<td>SHIFT-JIS</td>
<td>Basic</td>
</tr>
<tr>
<td>Simplified Chinese</td>
<td>CP936</td>
<td>Basic</td>
</tr>
<tr>
<td>All (Unicode)</td>
<td>UTF-8</td>
<td>ICU-UCA</td>
</tr>
</tbody>
</table>

**QAD SE Usage:**

QAD SE does not support Unicode (UTF-8 codepage) at this time.

**Progress + Oracle**

<table>
<thead>
<tr>
<th>Language/Region</th>
<th>PSC Code Page</th>
<th>PSC Collation</th>
<th>Oracle DB Character Set</th>
<th>NLS_LANG Environment Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Western European (includes US English)</td>
<td>ISO8859-1</td>
<td>Basic**</td>
<td>WE8ISO8859P1</td>
<td>&quot;DANISH_DENMARK.WE8ISO8859P1&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;DUTCH_NETHERLANDS.WE8ISO8859P1&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;FINNISH_FINLAND.WE8ISO8859P1&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;FRENCH_FRANCE.WE8ISO8859P1&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;GERMAN_GERMANY.WE8ISO8859P1&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;ITALIAN_ITALY.WE8ISO8859P1&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;MEXICAN_SPANISH_MEXICO.WE8ISO8859P1&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;BRAZILIAN_PORTUGUESE_BRAZIL.WE8ISO8859P1&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;AMERICAN_AMERICA.WE8ISO8859P1&quot; (and others as appropriate for the application client)</td>
</tr>
<tr>
<td>Eastern European</td>
<td>1250</td>
<td>Basic**</td>
<td>EE8ISO8859P2</td>
<td>&quot;CZECH_CZECHOSLOVAKIA.EE8ISO8859P2&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;HUNGARIAN_HUNGARY.EE8ISO8859P2&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;POLISH_POLAND.EE8ISO8859P2&quot;</td>
</tr>
<tr>
<td>Simplified Chinese</td>
<td>CP936</td>
<td>Basic**</td>
<td>ZHS16GBK</td>
<td>&quot;SIMPLIFIED_CHINESE_CHINA.ZHT16GBK&quot;</td>
</tr>
</tbody>
</table>
### All (Unicode)
- **UTF-8**
- **Basic**
- **AL32UTF8**
- "AMERICAN_AMERICA.AL32UTF8"
- "FRENCH_FRANCE.AL32UTF8"
- "POLISH_POLAND.AL32UTF8"
- "SIMPLIFIED CHINESE_CHINA.AL32UTF8"
(And so on depending on the language and region settings the clients use)

**The Progress DataServer for Oracle supports only the Progress collation table named Basic.**

### Progress + SQL Server

<table>
<thead>
<tr>
<th>Language/Region</th>
<th>PSC Code Page</th>
<th>PSC Collation</th>
<th>MSSQL Encoding Scheme</th>
<th>MSSQL Collation</th>
</tr>
</thead>
<tbody>
<tr>
<td>US English</td>
<td>ISO8859-1</td>
<td>Basic</td>
<td>CP1</td>
<td>SQL_Latin1_General_CP1_CI_AS</td>
</tr>
<tr>
<td>Western European</td>
<td>ISO8859-1</td>
<td>Basic</td>
<td>CP1</td>
<td>Latin1_General_CI_AS</td>
</tr>
<tr>
<td>Eastern European</td>
<td>1250</td>
<td>Czech, Hungarian, Polish ...</td>
<td>CP1250</td>
<td>Czech_CI_AS, Hungarian_CI_AS, Polish_CI_AS ...</td>
</tr>
<tr>
<td>Japanese</td>
<td>SHIFT-JIS</td>
<td>Basic</td>
<td>Japanese</td>
<td>Japanese_CI_AS</td>
</tr>
<tr>
<td>Simplified Chinese</td>
<td>CP936</td>
<td>Basic</td>
<td>PRC</td>
<td>Chinese_PRC_CI_AS</td>
</tr>
<tr>
<td>All (Unicode)</td>
<td>UTF-8</td>
<td>ICU-UCA</td>
<td>UCS-2</td>
<td>Choose appropriate collation for client via query:</td>
</tr>
</tbody>
</table>

**The Progress DataServer for MS SQL Server defaults to the Progress collation table named Basic, though others can be used.**
STD-0023 INPUT function on a variable referenced in multiple frames

Summary

Use 'frame' specification in INPUT function for a variable used in more than one frame.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0023</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

When Progress uses the INPUT function to reference the value of a variable without a frame reference, it will look for the first occurrence of this variable in the stack. The problem surfaces when there are two frames referencing the same variable. The problem occurs even if "DO WITH FRAME" statement is used to scope a variable.

Behaviors to look for:

- Compile Warnings mentioned above.
- "**PROMPT variable &lt;name&gt; should be used with INPUT prefix or ASSIGNED.""
- Typing in known values, but receiving "does not exist" type messages. i.e. typing in a valid part number only to get an "ITEM NUMBER DOES NOT EXIST" error message.

To remedy this problem, put a "frame" specification on the INPUT function, as specified in the example below.

Rationale

This standard helps programmers to avoid compile and run-time problems when a variable is used in more than one frame.

Examples

Right
In this program, even though "DO WITH FRAME m" is specified, the INPUT function is not able to resolve the frame reference.
STD-0024 Expanding text for longer translation

Summary

Translated versions of English text often require more space than the English original. The developer should always allow for displayable strings (such as field labels) to be longer in their translation into other languages.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0024</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Translatable Strings</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Most languages are less concise than English. Text strings that are displayed on screens or reports must therefore be at least 30% expandable beyond the English original. Also never limit a text strings to only 3 or fewer bytes.

General guidelines are:

<table>
<thead>
<tr>
<th>Original text length of English string</th>
<th>Expansion requirement in %</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 10 characters</td>
<td>30</td>
</tr>
<tr>
<td>0 - 10 characters</td>
<td>100</td>
</tr>
</tbody>
</table>

Rationale

Translated text that expands past the boundaries of an enclosing label creates a bad user interface.

Examples

Right

```plaintext
form
addr1   label "Old Address" colon 15
name1   no-label
addr2   label "New Address" colon 15
name2   no-label
with frame a side-labels width 80 attr-space.
```

Wrong

```plaintext
form
addr1   label "Old Address" colon 12
name1   no-label
addr2   label "New Address" colon 12
name2   no-label
with frame a side-labels width 80 attr-space.
```
STD-0025 Avoid string concatenation in Oracle

Summary

String concatenation can cause incomplete results when retrieving records in Oracle.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0025</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Data Handling Logic, Table Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Avoid concatenating fixed length data into a single database field or variable used for record retrieval with a WHERE clause. Typically such data is padded with trailing blanks to complete the fixed length in these instances. Progress will ignore the trailing blanks, but Oracle will interpret them literally, looking for a record that also contains trailing blanks when using that portion of the field/variable for record retrieval using the WHERE clause. For example, Progress will interpret "abcd " as "abcd", whereas Oracle will use "abcd ". Oracle will execute the retrieval statement but will not return any records in such instances.

The same problem can also occur when a literal is concatenated on the end of a database field or variable for use in conjunction with the WHERE clause.

When these situations cannot be avoided, use the RIGHT-TRIM function to strip off the trailing blanks in the field/variable to be used for finding a record with the WHERE clause. (The TRIM function will strip off leading and trailing blanks. Only trailing blanks should be removed; therefore the RIGHT-TRIM function is used.) Oracle requires special coding considerations for strings with trailing blanks to correctly find records in the database.

Examples

Right

Example 1 (concatenated data in a single database field):

```sql
assignacct = "1234"
c = "10"
substring(tx2__qad01, 1, 8) = acct
substring(tx2__qad01, 9, 4) = cc.

/* After the above code, tx2__qad01 contains "1234 10". */

/* FIND THE ACCOUNT RECORD FOR "1234" */
for first ac_mstr where ac_code = right-trim(substring(tx2__qad01, 1, 8))
no-lock end.
```

Example 2 (concatenated literal in a WHERE statement):
This solution is correct but will cause additional processing overhead if the concatenation occurs in a loop (i.e., the concatenation is done for each iteration of the loop). Instead, concatenate the literal into a local variable and use the local variable in the WHERE clause.

local_var = right-trim(group) + "_00".
for each pt_mstr where pt_group = local_var no-lock: end.

Wrong

Example 1 (concatenated data in a single database field):

assign
acct = "1234"
c = "10"
substring(tx2__qad01,1,8) = acct
substring(tx2__qad01,9,4) = cc.

/* After the above code, tx2__qad01 contains "1234 10" */

/* FIND THE ACCOUNT RECORD FOR "1234" */
for first ac_mstr where ac_code = substring(tx2__qad01,1,8) no-lock end.

This code will fail in Oracle because the first 8 characters of tx2__qad01 include 4 trailing blanks. No trailing blanks occur in the ac_mstr record for ac_code "1234".

Example 2 (concatenated literal in a WHERE statement):

for first pt_mstr where pt_group = group + "_00" no-lock: end.
STD-0026 Don't build strings to be displayed by concatenating

Summary

Don't build labels and messages in the code by concatenating strings, because it is impossible to translate them. Labels that are formed by joining together individual text strings are untranslatable. No two languages have quite the same syntax, and by building a single label from more than one string, the translator of that label is forced to use English syntax. In most cases, English syntax works ONLY for English.

ID  STD-0026
Version  0
Language  Progress
Status  Published
Categories  Character UI Apps, Translatable Strings
Applicability  All development activities

Description

By building a single label from more than one string, the translator of that label is forced to use English syntax. In most cases, English syntax works ONLY for English. German, for example, likes occasionally to split verbs into two parts and throw one at the beginning of the sentence and one at the end. French, as another example, likes to have its adjectives behind the nouns modified. Turkish, as another example, shuns all free-standing prepositions.

Rationale

This sort of "splitting" of labels forces English syntax on the translators. A label should be a coherent, stand-alone string. Messages should be complete in themselves (and should be put in msg_mstr). Keep in mind that MFG/PRO is translated into more than twenty languages, each of which has its own rules for forming sentences.

Examples

Right

```
define variable msgString as character no-undo.

/* Message 3726 is "Must be in the range # to # or blank" */

{ pxmsg.i
  &ERRORLEVEL = 1
  &MSGNUM = 3726
  &MSGARG1 = string(first_code)
  &MSGARG2 = string(last_code)
  &MSGBUFFER = msgString
}

put unformatted msgString skip.
```

Wrong

```
put unformatted "Must be in the range " + string(first_code) + " to " + string(last_code) + " or blank" skip.
```
STD-0028 Control File Labels

Summary
Do not use “Default” in control file labels.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0028</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Translatable Strings, DB Fields</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
Field labels for control file fields should NOT have the word “Default” added to them, even if they are used to initialize new records. Use the same label that is used in other programs for a field of that type. Specify that the field is used as a default value in the field help.

Rationale
The word ‘Default’ is usually inappropriate when attached to control file fields, since these values may not always, or do not generally contain default values. The objection to the use of ‘Default’ for the label is that you end up with multiple fields in the control table that all have ‘Default’ in their label and it doesn’t add value and is repetitious. The entire table contains fields that are control values or default values or turn on and off certain behaviors and the word ‘Default’ isn’t helpful.

See Also
STD-0029 Field Labels - Do not include punctuation marks
STD-0030 Label Names
STD-0037 Abbreviations for Field Labels
**STD-0029 Field Labels - Do not include punctuation marks**

**Summary**

Punctuation marks are NOT to be used in field labels.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0029</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Translatable Strings, DB Fields</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

**Description**

Punctuation marks are NOT to be used in field labels. Especially avoid the use of question marks “?” in the labels.

**Rationale**

In many cases we follow the field label with a colon. If we have punctuation at the end of a label, the label will not appear correctly on the user interface when followed by a colon.

**Examples**

**Right**

```
Allow Mod/Del When Shipped:
```

**Wrong**

```
Allow Mod/Del When Shipped?:
```

**See Also**

- STD-0028 Control File Labels
- STD-0030 Label Names
- STD-0037 Abbreviations for Field Labels
STD-0030 Label Names

Summary
Give Labels a Full and Complete Name.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0030</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Translatable Strings, DB Fields</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
Give Labels a Full and Complete Name. The side-label (LABEL attribute) of all fields should be specified. The column-label (COLUMN-LABEL attribute) of a field should be left undefined (?) if it is identical to the side-label.

Rationale
You cannot translate words like “Line” into ideographic languages such as Chinese. The translator needs to know whether we are talking about Product Line, Sales Order Line, Production Line.

Examples
For schema column-labels you can use Line if the label is complete as that will tell the translator to basically translate the column-label the same way as the label, though perhaps somewhat abbreviated. However, if you think about hardcoding the label “Line”, don't. Instead, hardcode the whole label such as “Product Line”, “Sales Order Line”, etc. or an abbreviation (“Prod Line”, “SO Line”) which enables the translator to make the proper distinction and ultimately the correct translation.

See Also
STD-0028 Control File Labels
STD-0029 Field Labels - Do not include punctuation marks
STD-0037 Abbreviations for Field Labels
STD-0031 Report - Page Limits and Printing Interruption

Summary

The proper use of mfrpchk.i in report and inquiry programs enables page limit termination of printed output when reports are sent to printers with pre-defined page limits. It also allows user termination of report output to a terminal display device when the report is paused on a ‘Press spacebar to continue’ prompt. Use mfrpchk.i to control page limits and termination during printing.

ID | STD-0031
---|---
Version | 0
Language | Progress
Status | Published
Categories | Reports/Inquiries
Applicability | MFG/PRO Reports

Description

MFG/PRO has the ability to limit the number of report pages printed to a specific printer definition. This capability is set up in the printer definition using the "Lines / Page" and the "Max Pages" fields. The mfrpchk.i include file contains the logic necessary to act on the maximum pages setting defined for the printer. To function properly, this include file needs to be placed in the source code within each iterating block of the report that contains a display statement. It also needs to exist outside of each such block to pass the termination command to the outermost iterating code segment. This include file also holds the logic required to deal with the 'F4' key correctly (termination request) for reports that are sent to a terminal display device. It is to be noted that mfdeclre.i include file needs to exist in the same program as mfrpchk.i because of the shared variables used for page counting.

Rationale

The intent of this standard is to describe how to halt printed output for ‘page limited printers’ and to cancel a ‘paused’ report to terminal by using the F4 key in a consistent way.

Examples

Right

Place the include file inside the innermost loop containing a display statement and then outside of that loop and other nested loops. Use the suppress warning option for all but the outermost instance if the exit label option is not used.

There are two ways to exit a report using mfrpchk.i, by exiting each iterating loop explicitly or by exiting a larger labeled loop by name:

Option 1

```c
loopa:
    for each...
    for each ....
    for each ...
    display ...{mfrpchk.i &warn=no}
end.

{mfrpchk.i &warn=no}
end.{mfrpchk.i}
end.
```

The first and second instances of mfrpchk.i suppress the "report terminated" and "max page reached" messages while the third instance displays these messages as necessary. Without the warning suppressions, these
messages would display at each level from the point where the termination is triggered out to each subsequent level.

**Option 2**

```plaintext
loopa:
  for each...
  for each ....
  for each ....
  display ...{mfrpchki &label=loopa }
  end.{mfrpchki &label=loopa}
  end.
{mfrpchki &label=loopa}
end.
```

These instances of mfrpchki will completely exit loopa when the page limit is reached (or F4 is hit during a ‘report to terminal pause condition’). In this usage, the ‘suppress warning’ option is not used since only one of the include files will be acted upon (once one of them is activated, the other ones will be skipped).
STD-0032 Progress Schema Triggers

Summary

Included in this document are some standards for the use of PROGRESS schema triggers.

Triggers are not to be used in the QAD Financials schema.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0032</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Tables</td>
</tr>
<tr>
<td>Applicability</td>
<td>PROGRESS schema triggers</td>
</tr>
</tbody>
</table>

Description

1. Program naming convention
The naming convention for schema trigger source files is:

<table_prefix>[number][r]<c|w|d>.t

where:

<table>
<thead>
<tr>
<th>&lt;table_prefix&gt;</th>
<th>The prefix from the table name (e.g. 'gl' for table 'gl_ctrl')</th>
</tr>
</thead>
<tbody>
<tr>
<td>[number]</td>
<td>A number that may be inserted if there is a naming conflict with other source files (see below)</td>
</tr>
<tr>
<td>[r]</td>
<td>Signifies a (R)eplication trigger</td>
</tr>
<tr>
<td>&lt;c</td>
<td>w</td>
</tr>
</tbody>
</table>

If the resulting name exceeds the maximum defined in STD-0310: Progress source code file names in MFG/PRO, the [table_prefix] may be truncated as needed as long as the resulting file name is still unique. When creating a new schema trigger for a table, check to see if any schema trigger source files are already in existence for the table. If so, use the [number] if any that was used in these existing names. A [number] is necessary if there are any conflicting '.t', '.p' or '.w' source code files because all source code files regardless of the file extension will become .r files when compiled. A conflict with a ".t" suffix would be possible if there are two tables with the same prefix (e.g. rqm_mstr and rqm_det). Whenever possible, the same [number] should be used for all trigger files for a particular table.

2. Must be shippable and executable as part of base module
As schema triggers are referenced by the database schema, they should be included in all MFG/PRO releases even if the desired application functionality is not shipped. Any non-base-module functionality should therefore be placed in other ".p" programs executed from the trigger programs.

3. Must not include any user interface or other translatable components
As triggers are not called using gprun.i, they can't be released and stored under language-dependent subdirectories like the rest of the MFG/PRO code. For this reason, they shouldn't include any translatable items such as labels, messages, help, etc., or user interface elements. In general, programs called by schema triggers should also not contain such elements without very good reasons.

4. Move functionality-specific code to sub-procedure
Because a trigger executes for all events of one type (writes for example) for all records in the database table, logic that will not always apply should be moved to a sub-procedure and called conditionally. This is especially important for module-specific logic that should be moved to a sub-procedure and called using gprunmo.i specifying the module name. The trigger file should be written with the expectation that multiple unrelated enhancements will over time make use of the same triggers for diverse purposes.

5. Must be short and terminate with minimal database I/O
Because schema triggers impose additional system overhead on all update activity to their associated tables, they should always ‘fall through’ very quickly, typically after a conditional statement, when their functionality is not needed. In particular, the database I/O performed in such cases should be minimal.

6. **FIND and ASSIGN schema triggers should not be used**
Because of their potentially high performance impact, the PROGRESS FIND and ASSIGN trigger types, which are field- rather than table-sensitive, should not be used in standard product without special permission of the QAD Architects.

7. **Do not use mfdeclre.i in trigger (.t) programs.**
There are very few shared variables needed in most trigger source files, the most common being global_user_lang_dir needed for calling a sub-procedure using gprun.i. Use an existing shared variable include file that contains the minimum number of shared variables that will meet the needs of the trigger file. This will reduce the compiled size of the trigger and speed execution. A sub-procedure that is called from the trigger file can use a different shared variable include file, even mfdeclre.i if needed.

Note that when shared variables are used in triggers, this will produce run-time error when data changes are initiated from the Progress Editor. Replication triggers must be designed to execute without errors even when changes are made from the Progress Editor. This can be accomplished by defining the shared variables as “new global” along with conditional logic for when the variables are not populated (e.g. from a Progress Editor session).

8. **Avoid complex update processing**
In general, schema triggers should not be used for extensive table updates in order to avoid side-effects that might reduce the maintainability or reliability of MFG/PRO. Typical natural uses for schema triggers include populating the main OID field, data extraction/replication, audit trails, and the enforcement of referential integrity (e.g., automatically deleting all detail records associated with a deleted header). Other kinds of applications should be discussed with the QAD Architect team before using them in standard product.

**Rationale**

Schema triggers, when used inappropriately, will have huge impact on performance, maintainability, run time characteristics etc.

**Examples**

**Right**

**Code**

1. **Program naming convention**
Examples:
- “adc.t” is the create trigger for Address Master (ad_mstr).
- “sodw.t” is the write trigger for Sales Order Detail (sod_det).
- “mrpd.t” is the delete trigger for Material Requirements Planning Detail (mrp_det).
- “knbadcrw.t” is the replication write trigger for Kanban Average Demand Calculation History (knbadc_hist).
- “wodrd.t” is the replication delete trigger for Work Order Detail (wod_det).
- “rqm2rw.t”, is the replication write trigger for Requirement Detail (rqm_det). There was a naming conflict with an existing source file (rqmrw.p) and another table has the same prefix (rqm_mstr) that already has triggers defined using [number]=1. All triggers defined for rqm_det should use [number]=2.

2. **Must be shippable and executable as part of base module**
Example:
Not applicable

3. **Must not include any user interface or other translatable components**
Example: sodd.t

```progress
if not can-find(first whl_mstr where whl_mstr.whl_act) then leave.
```

4. **Move functionality-specific code to sub-procedure**
Example: sodw.t
Note: The following trigger delegates Data Warehouse related functionality to a sub-program called ‘wisodw.p’ instead of coding the logic in the main trigger itself.
5. Must be short and terminate with minimal I/O when not used
Example: cmw.t
Note: In this example, if Logistics is not in use as determined by the can-find statement, then processing ends.

```plaintext
/* Logistics Interface processing */
LG:
do:
/* If there is no logistics application ID, then do not process */
if can-find(first lgs_mstr no-lock
    where lgs_mstr.lgs_domain = global_domain
    and lgs_mstr.lgs_app_id <> "")
then do:
/* Did the credit limit or balance change? */
if old_cm_mstr.cm_balance <> cm_mstr.cm_balance or
    old_cm_mstr.cm_cr_limit <> cm_mstr.cm_cr_limit or
    old_cm_mstr.cm_curr <> cm_mstr.cm_curr
then do:
/* Yes, call logistics to process */
gprunmo.i &module="LG"
    &program="lgarex.p"
    &param=""(input recid(cm_mstr), output nPublished)"
end.
end.
end.
/* LG */
```

6. FIND and ASSIGN schema triggers should not be used
Example: Not applicable

7. Do not include mfdeclre.i in trigger (.t) programs.
Example: ardw.t. This write trigger needs two global variables defined: global_domain for can-find, and global_user_lang_dir for use with gprunmo.i. These two shared variables along with global_userid are defined in mfvar.i. The called program, lgpyex.p, includes mfdeclre.i.

```plaintext
TRIGGER PROCEDURE FOR WRITE OF ARD_DET OLD BUFFER OLD_ARD_DET.
/* Supply the old table values to lgpyex to determine differences */
{mfvar.i}
define new shared temp-table tt_ard_det no-undo like ard_det.

/* Logistics Interface processing */
LG:
do:
/* If Logistics is in place, process */
if can-find (first lgs_mstr where lgs_domain = global_domain and
    lgs_app_id > "")
then do:
    buffer-copy old_ard_det to tt_ard_det.
/* Call subprogram to publish changed status for Logistics */
gprunmo.i &module="LG"
    &program="lgpyex.p"
    &param=""(input recid(ard_det))"
if available (tt_ard_det) then delete tt_ard_det.
end. /* LG */
```

8. Avoid complex update processing
Example: Not applicable
Wrong

Code

1. Program naming convention

Examples:

"pt.t" is the create trigger for pt_mstr - the file name does not have {c|w|d} character to indicate whether it is a create, write or delete trigger.

"sod.p" is the delete trigger for sod_det - the file name ends with ".p"; the file name extension should be ".t"

"rqmw.t" is the write trigger for rqm_mstr - this file name should not be used because other trigger files have already been defined for rqm_mstr using [number] 1. This trigger file should be named "rqm1.t".

"sod.p" is the replication delete trigger for so_mstr - the file name needs to include an [r] indicating (R)eplication. It should be "sord.p".

2. Must be shippable and executable as part of base module

Example: Not applicable

3. Must not include any user interface or other translatable components

Example: sod.t

```plaintext
if not can-find(first whl_mstr where whl_mstr.whl_act)
then message "Warehouse Master not found".
```

4. Move functionality-specific code to sub-procedure

Example: sodw.t

Note: The following trigger has logic specific to Data Warehouse module; Therefore, it should be moved to a sub-program which deals with Data Warehouse functionality, as illustrated in the "Right - Code" section, example #4.

```plaintext
/* If the line being added/changed has a positive quantity ordered, then */
/* create an add/change transaction if there is a picked quantity. */

if sod_det.sod_qtyOrd >= 0 then do:
    for each lad_det /*J0KF*/ no-lock
        where lad_det.lad_domain = global_domain
            and lad_det.lad_dataset = "sod_det"
            and integer(lad_det.lad_line) = sod_det.sod_line
            and lad_det.lad_part = sod_det.sod_part
            and lad_det.lad_site = sod_det.sod_site
        break by lad_det.lad_loc:
/* More module-specific code */
end.
```

5. Must be short and terminate with minimal database I/O when not used

Example: sodw.t

Note: In this example, none of this processing is necessary if Data Warehousing is not being used. This could be determined by a simple can-find statement and this entire block should only be conditionally executed.
do:
  /* If the order quantity is being changed from - to +, then create a delete transaction first */
  if wf_old_sod_det.sod_qty_ord < 0
    and sod_det.sod_qty_ord >= 0
  then do:
    w-file = "wisodd.p".
    {gprunmo.i &module="WI" &program="wisodd.p"}
  end.
/* If the line being added/changed has a negative quantity ordered, then create an add/change transaction */
  if sod_det.sod_qty_ord < 0
  then do:
    w-file = "wisodw2.p".
    {gprunmo.i &module="WI" &program="wisodw2.p"}
    leave WI.
  end.
/* If the line being added/changed has a positive quantity ordered, then create an add/change transaction if there is a picked quantity */
  if sod_det.sod_qty_ord >= 0
  then do:
    w-file = "wisodw.p".{gprunmo.i &module="WI" &program="wisodw.p"}
    end.
  end.

6. FIND and ASSIGN schema triggers should not be used
Example: Not applicable

7. Do not include mfdeclre.i in trigger (.t) programs.
Example: mfdeclre.i is included in this trigger file even though only two shared variables are needed (global_userid, global_user_lang_dir). mfvar.i should have been used instead.

TRIGGER PROCEDURE FOR WRITE OF ARD_DET OLD BUFFER OLD_ARD_DET.

/* Supply the old table values to lgpyex to determine differences */
{mfdeclre.i}
define new shared temp-table tt_ard_det no-undo like ard_det.
/* Logistics Interface processing */
LG:
do:
  /* If Logistics is in place, process */
  if can-find (first lgs_mstr where lgs_domain = global_domain and lgs_app_id > "")
    then do:
      buffer-copy old_ard_det to tt_ard_det.
      /* Call subprogram to publish changed status for Logistics */
      {gprunmo.i &module="LG" &program="lgpyex.p" &param=""(input recid(ard_det))""
        if available (tt_ard_det) then delete tt_ard_det.
        end.
    end. /* LG */

8. Avoid complex update processing
Example: Not applicable

See Also

STD-0326 Every qaddb table must have a create trigger (eB3 and up)
STD-0310 Progress source code file names in MFG-PRO
STD-0034 Never set the mfguser variable

Summary
The variable mfguser should always be set by initialization during the MFG/PRO login process. It should never be set by any other program.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0034</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Data Handling Logic</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
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 MFG/PRO session, typically work-file records. These records usually have a xx_user field in the table which is assigned the value of mfguser when records are created. Mfguser is also used to select records when deleting records before processing begins and after processing ends. Note! These work-files should not be used in any new development, instead temp-tables should be used which eliminates the need for mfguser.

When mfguser is needed this is how it should be used:

- The variable mfguser should be always be set by initialization during the MFG/PRO login process. Mfguser is a shared variable declared in mfdeclre.i and is set by mf1a.p upon initial entry into the MFG/PRO main menu.
- Mfguser should never be reset by any other program.
- Mfguser should not be used as a local variable name; this would be confusing to maintenance personnel.
- When a program is executed which will require access to mfguser, the program should only be executed via the MFG/PRO menu in order to guarantee a unique value for mfguser (i.e., not from a stand-alone Progress session).

Rationale
It is crucial to the proper operation of many programs that this variable is guaranteed a value unique to the current Progress session.

Examples

Right

Code
```plaintext
{mfdeclre.i}
define variable my_var like mfguser.
my_var = mfguser.
```

Wrong

Code
/* never define mfguser locally */
define variable mfguser as character no-undo.

/* never assign your own value to mfguser */
mfguser = "abc1234".
STD-0036 Handling of Directory Names and Paths

Summary

The include file gpdirpre.i can be used to help process a path name and strip out directories or paths based on the directory character ("/" or ", depending on operating system).

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0036</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Data Handling Logic, Operating System</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

If you need to process a path name and strip out directories or paths based on the directory character, the include file gpdirpre.i declares and sets a character variable, dir_prefix, for whichever operating system on which the Progress session is running. This can also be used for constructing a pathname.

Rationale

Isolates OS-dependency with respect to pathname convention to a single location.

Example

Right

Code

```progress
define variable path_name as character no-undo.
define variable sub_dir as character no-undo.
define variable file_name as character no-undo.
{gpdirpre.i}
...
path_name = sub_dir + dir_prefix + file_name.
```

Wrong

Code

```progress
define variable path_name as character no-undo.
define variable sub_dir as character no-undo.
define variable file_name as character no-undo.
... /* the following may cause problems on non-Unix systems */
path_name = sub_dir + "/" + file_name.
```
STD-0037 Abbreviations for Field Labels

Summary
Use only approved abbreviations for field labels.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0037</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, DB Fields, Translatable Strings</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

<table>
<thead>
<tr>
<th>Description</th>
<th>GL Account Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account (Acct is acceptable)</td>
<td>GL Account Code</td>
</tr>
<tr>
<td>Amt</td>
<td>Amount</td>
</tr>
<tr>
<td>CC (Cost Ctr is acceptable)</td>
<td>Cost Center</td>
</tr>
<tr>
<td>Cur</td>
<td>Currency</td>
</tr>
<tr>
<td>Curr</td>
<td>Current</td>
</tr>
<tr>
<td>Disc</td>
<td>Discount</td>
</tr>
<tr>
<td>Enty</td>
<td>Entity</td>
</tr>
<tr>
<td>Iss</td>
<td>Issue</td>
</tr>
<tr>
<td>Ln</td>
<td>Line</td>
</tr>
<tr>
<td>Max</td>
<td>Maximum</td>
</tr>
<tr>
<td>Min</td>
<td>Minimum</td>
</tr>
<tr>
<td>Nbr</td>
<td>Number</td>
</tr>
<tr>
<td>PO</td>
<td>Purchase Order</td>
</tr>
<tr>
<td>Qty</td>
<td>Quantity</td>
</tr>
<tr>
<td>Rcpt</td>
<td>Receipt</td>
</tr>
<tr>
<td>SO</td>
<td>Sales Order</td>
</tr>
<tr>
<td>Sub-Acct (Sub is acceptable)</td>
<td>GL Sub Account Code</td>
</tr>
<tr>
<td>UM</td>
<td>Unit of Measure</td>
</tr>
<tr>
<td>WIP</td>
<td>Work in Process</td>
</tr>
<tr>
<td>WO</td>
<td>Work Order</td>
</tr>
</tbody>
</table>

Rationale
Different abbreviations require additional translation and create inconsistencies.

See Also
STD-0028 Control File Labels
STD-0029 Field Labels - Do not include punctuation marks
STD-0030 Label Names
STD-0039 Use of 'pause 0 before-hide' statement - pause 0 is preferred

Summary

Use of "pause 0 before-hide" statement may cause UI problems - usage of "pause 0" is preferred

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0039</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Attempt to avoid the use of "pause 0 before-hide". Use "pause 0" instead. Use of "pause 0 before-hide" sets a global Progress condition which can cause problems for Field and Procedure Help as well as multi-page inquiries which rely on the default behavior of hiding frames which Progress uses. Field Help Display will not work properly with programs that contain a "pause 0 before-hide" statement. The "before-hide" option will display the Help frames in a non-stop manner and then clear the final Help frame from the screen before returning to the calling program. In many instances, simply removing the "before-hide" option from the statement (and leaving the "pause 0" intact) will correct this problem. Try this solution first; if the help text is still not displaying correctly (or other problems appear) the structure of your program may need to be altered.

Sometimes it may be necessary to override Progress’ intrinsic frame hiding and viewing in order to avoid the "Press space bar to continue" message. If the "pause 0 before-hide" is invoked prior to any user interaction commands (prompt-for/set/update, etc.) then it is safe to use. However, because future code modifications may add user interaction commands which will then cause the "pause 0 before-hide" to work incorrectly, use of the "pause 0 before-hide" is discouraged even in this situation.

Examples

Right

Code

```plaintext
pause 0.
```

Wrong

Code

```plaintext
pause 0 before-hide.
```
STD-0040 Ensure that a user can respond to 'Press space bar to continue' with either the spacebar or end-key (F4-esc)

Summary

Users often respond with the End key (F4/Esc) when prompted to "Press space bar to continue". When they do so, the program behavior should be the same as if they responded with the space bar.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0040</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Users have the habit of pressing End-key (F4/esc) instead of the space bar at "Press space bar to continue" prompts. This action should not have unintended results, e.g. the unintended backing out of transactions. The best choice is to avoid explicit pause statements whenever possible and practical, but if the pause is necessary, this problem can be solved by surrounding the PAUSE statement with a DO statement as shown below. The DO statement should also be conditioned on "batchrun = no" as all PAUSE statements should be ignored if the program is executing in CIM (batchrun = yes) mode.

When an explicit pause is required, the use of the end-key should not be ambiguous, i.e., there should not be two possible responses: space to continue and F4 to back out.

Rationale

When users press end-key instead of the space bar at "Press space bar to continue" prompts, it is generally not the users intent to back out the transaction. Thus, this action should not have unintended results, such as inadvertently backing out an entire transactions and causing the user to re-enter data. When the program backs out of transactions when the user responds that way, the program behavior is generally reported as a bug. Although technically this action may be considered a user error, it is important to write robust software that attempts to prevent user from making predictable errors.

Examples

Right

Code

```plaintext
if not batchrun then do on endkey undo, leave: 
  pause.
end.
```

Wrong

Code

```plaintext
pause.
```

See Also
STD-0052 CIM Interface Compatibility
STD-0041 Use of user fields in MFG-PRO is NOT allowed

Summary
Do NOT store or maintain data in user-reserved fields.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0041</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields, Field Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Programs are NOT permitted to store, maintain, or make use of user-reserved fields.

Every MFG/PRO table definition contains user-reserved fields such as:
- "_user"
- "_chr"
- "_dte"
- "_dec"
- "_log"

These user-reserved fields are designed to allow customers to store relevant information in existing tables without having to make schema additions. User-reserved fields are unavailable for QAD use. In no circumstance should user-reserved fields be utilized by QAD developers.

Rationale

QAD could easily corrupt customer’s data by making reference to user-reserved fields. An ECO or new release which contains programs that update data in user-reserved fields could over-write existing customer data.

Example

For example, a need arises to add a second bank code to the A/R account. The second bank should be maintained as part of the Accounts Receivable (ar_mstr) table, but the schema is frozen and no new fields can be added at this time. We notice that the ar_mstr contains fields named ar_user1, ar_user2, ar_chr01, ar_chr02, etc., as well as QAD-reserved fields ar_qadc01, ar_qadc02, etc.

The correct approach is to determine if QAD-reserved field ar_qadc01 or ar_qadc02 is available to store the second bank code. (A grep of all programs should show if these fields are being used by other programs.) If available, use one of these QAD-reserved fields until a properly named field is added to the schema in the next release (See STD-0050, Required Table Fields in the “See Also” section below).

If neither of these fields are available, then we would use either a qad_wkfl record (for pre-eB2 releases) or a qtbl_ext record (for eB2 and newer) to store the new bank code in until the properly named field was added to the schema (for pre-eB2, see STD-0067 Proper Use of qad_wkfl, and STD-0305 Extending frozen schema tables to provide extra QAD fields (eB2 and up) in the “See Also” section below).

We should not use the user-reserved fields ar_user1, ar_user2, ar_chr01, etc.

Right

The following examples show solutions for MFG/PRO versions prior to eB2 (eB, 9.0, 8.6, etc.). For eB2 and newer releases (eB2, eB2.1, QAD SE, QAD EE), see examples in STD-0305 Extending frozen schema tables to provide extra QAD fields (eB2 and up).
Code

When ar_qad01 field IS available:

```sql
/* ********** Begin Translatable Strings Definitions ********** */
&SCOPE DEFINE pgm_p_1 "Bank2"
/* ********** End Translatable Strings Definitions ********** */

/* THE NEXT LINE WILL BE REMOVED WHEN ar_bank2 IS IN THE DB */
define variable ar_bank2 as character format "x(2)" label {&pgm_p_1}.

form
    ar_bank   colon 20
    ar_bank2  colon 20
with frame bank width 80.

/* THE NEXT LINE WILL BE REMOVED WHEN ar_bank2 IS IN THE DB */
ar_bank2 = ar_qad01.

update
    ar_bank
    ar_bank2
with frame bank.

/* THE NEXT LINE WILL BE REMOVED WHEN ar_bank2 IS IN THE DB */
ar_qad01 = ar_bank2.
```

OR - if ar_qad01 field is NOT available:

```sql
```

QAD Development Standards
Wrong Code

form
  ar_bank  colon 20
  ar_user1 colon 20 label "Bank 2" format "x(2)"
with frame bank width 80.

update
  ar_bank
  ar_user1
with frame bank.

See Also
STD-0050 Required Table Fields
STD-0051 Providing for Extra Customer Fields
STD-0042 Obsoleting Fields
STD-0150 Local Variable Naming Conventions
STD-0305 Extending frozen schema tables to provide extra QAD fields (eB2 and up)
STD-0042 Obsoleting Fields

Summary

Observe fields in QAD product schema by changing its name and other attributes as listed below.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0042</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>3</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

If you are obsoleting a field, change the name of the field but not the data type. Choose the allowed naming conventions from the list for each data type below.

- \texttt{xx___qadc##} character
- \texttt{xx___qadd##} decimal
- \texttt{xx___qadi##} integer
- \texttt{xx___qadl##} logical
- \texttt{xx___qadt##} date

For instance, so_mstr could have so___qadc03, so___qadc04, so___qadi02, etc.

A double underscore is used in the field name after the table prefix.

The numbers used should be sequentially assigned and should not be re-used.

Replace the original field attributes with those listed in the table below:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Format</th>
<th>MAX-WIDTH*</th>
<th>ValExp</th>
<th>ValMsg</th>
<th>Label Column Label</th>
<th>Initial</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx___qadc##</td>
<td>x(24)</td>
<td>80</td>
<td></td>
<td>?</td>
<td>**</td>
<td>?</td>
<td>QAD reserved field.</td>
</tr>
<tr>
<td>xx___qadd##</td>
<td>-&gt;&gt;&gt;&gt;,&gt;&gt;&gt;,&gt;&gt;9.9&lt;&lt;&lt;&lt;&lt;</td>
<td>38</td>
<td>**</td>
<td>?</td>
<td>&quot;0&quot;</td>
<td></td>
<td>QAD reserved field.</td>
</tr>
<tr>
<td>xx___qadi##</td>
<td>-&gt;&gt;&gt;&gt;&gt;&gt;9</td>
<td>n/a</td>
<td>**</td>
<td>?</td>
<td>&quot;0&quot;</td>
<td></td>
<td>QAD reserved field.</td>
</tr>
<tr>
<td>xx___qadl##</td>
<td>yes/no</td>
<td>n/a</td>
<td>**</td>
<td>?</td>
<td>&quot;no&quot;</td>
<td></td>
<td>QAD reserved field.</td>
</tr>
</tbody>
</table>

\textbf{MAX-WIDTH} should be increased to the value shown, but never decreased.
Rationale

We need to ensure obsolete fields do not adversely impact the upgrade or conversion of a product's database schema.

It is not possible to just delete a field in the production system schema – the field may be necessary for the conversion process. To allow for this activity, we rename the field. The Development Systems group creates a list of schema changes which are published with new releases to allow customers and vendors who may be using the field to be aware of the changes. Fields that have been renamed can then be re-used by some other functionality in a later release.

Fields need to be kept and not removed from the schema in order to maintain dump/load order. This is especially important for service pack upgrades and conversions where data would need to be moved from one database to another. Progress data reloads depend on the order of the fields, so that if data dumped from a table in one schema were to be loaded into the same table in which a field was dropped/deleted from the middle in a later schema, the load would most likely fail. Renaming obsolete fields prevents developers from having to write special dump/load routines in cases where a field has been dropped from a table because its data is no longer required. By simply changing a field's name to obsolete the field, the table's dump/load order is maintained and there is no need for developers to create special data movement programs for product service pack upgrades or conversions.

Furthermore, the time required to drop a field on Oracle and SQL Server is lengthy so we try to avoid this during conversions of products which use those database platforms in addition to Progress.

Example

Right

Code

```
RENAME FIELD "arc_cont_acct" OF "arc_ctrl" TO "arc__qadc03"
UPDATE FIELD "arc__qadc03" OF "arc_ctrl"
   DESCRIPTION "QAD reserved field"
   LABEL ?
   COLUMN-LABEL ?
   VALEXP ""
   VALMSG ""
   INITIAL ""
   FORMAT "x(24)"
   MAX-WIDTH 80
```

Wrong

Code

```
RENAME FIELD "arc_cont_acct" OF "arc_ctrl" TO "arc_not_used"
UPDATE FIELD "arc_not_used" OF "arc_ctrl"
   DESCRIPTION "NOT USED"
   LABEL "NOT USED"
   COLUMN-LABEL "N U"
```

See Also

STD-0057 Deleting Fields
STD-0307 Providing for Extra QAD Fields
STD-0043 Field Formats Allowed

Summary

Allowed Field Formats

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0043</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>3</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields, MFG/PRO Application</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Use the following formats when defining a field:

<table>
<thead>
<tr>
<th>Type</th>
<th>New Format</th>
<th>Decimals</th>
<th>Previous Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID Fields</td>
<td>&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;9.9&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Notes:</td>
<td>27 “&gt;”s + 9.9 + 9 “&lt;”s SQL-WIDTH = 38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GL Money</td>
<td>-&gt;&gt;&gt;,&gt;&gt;&gt;,&gt;&gt;&gt;,&gt;&gt;9.99&lt;</td>
<td>10</td>
<td>-&gt;&gt;&gt;,&gt;&gt;&gt;,&gt;&gt;&gt;,&gt;&gt;9.99</td>
</tr>
<tr>
<td>Doc Amount</td>
<td>-&gt;&gt;&gt;,&gt;&gt;&gt;,&gt;&gt;&gt;,&gt;&gt;9.99&lt;</td>
<td>10</td>
<td>-&gt;&gt;&gt;,&gt;&gt;&gt;,&gt;&gt;&gt;,&gt;&gt;9.99</td>
</tr>
<tr>
<td>Prices</td>
<td>-&gt;&gt;&gt;,&gt;&gt;&gt;,&gt;&gt;&gt;,&gt;&gt;9.99&lt;&lt;</td>
<td>10</td>
<td>-&gt;&gt;&gt;,&gt;&gt;&gt;,&gt;&gt;&gt;,&gt;&gt;9.99&lt;&lt;</td>
</tr>
<tr>
<td>Quantity</td>
<td>-&gt;&gt;&gt;,&gt;&gt;&gt;,&gt;&gt;9.99&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;</td>
<td>10</td>
<td>-&gt;&gt;&gt;,&gt;&gt;&gt;,&gt;&gt;9.99&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;</td>
</tr>
<tr>
<td>Date</td>
<td>99/99/99</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Year</td>
<td>9999</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical</td>
<td>yes/no</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Percentage</td>
<td>-&gt;9.9&lt;%</td>
<td>2</td>
<td>-&gt;9.99%</td>
</tr>
<tr>
<td>Codes</td>
<td>x(8)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phone</td>
<td>x(16)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Currency</td>
<td>x(3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Item Number</td>
<td>x(18)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>x(24)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UM</td>
<td>x(2)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Known Exceptions:

In some cases, a date format may be 99/99/9999, but only if the range of valid dates for the field can logically fall outside a single century – for example, employee birth dates.

Make sure you do not exceed Progress' maximums for the data type of the field in your format statements. For example, the maximum value of an integer is +/- 2,147,483,648 (about 2 billion). You can’t even use billions in the format because it isn’t 9 billion, so you must only allow hundreds of millions. Violations of this are typically found in...
dealing with currency amounts where it was assumed that only whole units would be needed, such as the customer's credit limit cm_cr_limit).

Examples

N/A

See Also

STD-0064 Character Field Format length limit
STD-0066 Do not use Packed Fields
STD-0100 Year fields - Format and initial value
STD-0333 Database fields with Extents are not supported
STD-0090 Schema field formats should support storage requirements for Oracle
STD-0301 Oracle MAX-WIDTH (SQL-WIDTH) must be specified on new database fields
STD-0351 Supported Progress Data Types
STD-0044 Definition And Acceptable Values For Logical Variables In Progress

Summary
Definition and acceptable values for logical variables in Progress.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0044</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps,Variable Definition,Translatable Strings</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
The initial value of a logical field should always be either yes or no in lower case. The value should NOT be whatever will fit in your format nor in quotes.

⚠️ Use "like mfc_logical" if the variable might be used in UI.

Rationale
Use of values other than yes or no will increase the translation problems by requiring the translation of non-standard values. In MFG/PRO, formats are translated but NOT their initial value. Use of "like mfc_logical" when defining the variable assures the consistency of field attributes and column labels.

Examples
For some instances of the format "Single/Multi" (whether for a field defined in the schema or a variable defined in some program) where you want the initial value to be "Single", you need designate the initial value to be yes, for "Multi" designate the initial value to be no.

Right

yes
no

Code

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

```define variable summary like mfc_logical format "Summary/Detail" initial yes no-undo.
/*Of course "Summary/Detail" would be a preprocessor label rather than a hard coded label in real code.*/
```

```define variable tmp_status like mfc_logical format "Open/Closed" initial yes no-undo.
/*Use yes or no without quotes with the INITIAL construct.*/
```

Wrong

Y
define variable valid_acct like mfc_logical initial "yes" no-undo.
/*"yes" would be translated.*/

define variable summary like mfc_logical format "Summary/Detail" initial "Summary" no-undo.
/*"Summary" would need to be translated.*/

define variable status as logical format "Open/Closed" initial "Open" no-undo.
/*Causes "Open/Closed" to be translated separately from "Open".*/
/*Results in extra translation cost and potentially different translation of the string in the two cases.*/

define variable tmp_status as logical format "open/closed" initial "Yes" no-undo.
/*Interprets "Yes" as a string to be translated, which it shouldn't.*/
STD-0045 Translatable hard-coded strings should appear only in a tokened area (pre-eB2)

Summary
Translatable hard-coded strings should appear only in a tokened area at the top of each file. In this tokened area, the strings are defined as preprocessors that are referenced within the code itself.

ID: STD-0045
Version: 1
Language: Progress
Status: Published
Categories: Character UI Apps, Translatable Strings
Applicability: All development activities (MFG/PRO versions eB and lower)

Description
Translatable hard-coded strings should appear only in a tokened area at the top of each file. In this tokened area, the strings are defined as preprocessors that are referenced within the code itself.

We do not use Progress's :U notation to indicate strings that are to remain untranslated.

Putting the labels in the token area provides a way to easily identify strings that need to be translated.

Examples
Example of a translatable hard-coded string definition area:

```c
/* ********** Begin Translatable Strings Definitions ********** */
&SCOPED-DEFINE test_p_1 "New Cust Address"
/* MaxLen:21 Comment: New Customer Address */
&SCOPED-DEFINE test_p_2 "Old Cust Address"
/* MaxLen:21 Comment: Old Customer Address */
/* ********** End Translatable Strings Definitions ********** */
```

Delineating the definition area
The opening line of a translatable strings definition area is:

```c
/* ********** Begin Translatable Strings Definitions ********** */
```

The closing line of a translatable strings definition area is:

```c
/* ********** End Translatable Strings Definitions ********** */
```

Definition format
Between the opening and closing lines of the definition area, all the quoted translatable strings in the file are assigned preprocessors. Each preprocessor is scoped-defined and each definition line begins with "&SCOPED-DEFINE" followed by the preprocessor variable name, followed by the quoted string:

```c
&SCOPED-DEFINE <preprocessor variable name> "<string>"
```
Naming convention of the preprocessors

The naming convention for the preprocessor variable name is:

\[ \text{<filename>} + "\_" + \text{<file extension>} + "\_" + \text{<ID number>} \]

where \text{filename} refers to the file the preprocessor appears in, file extension is "p","w" or "i", and ID number is any number which has not already been used by other preprocessors within the tokened area. An example of a preprocessor variable name, using the above convention used to define a string in the file adadcgp.p:

\[ \text{adadcgp\_p\_1} \]

Maximum length

On the line directly below each definition line is the "Max Length and Comment" line. This line begins and ends with comment marks (/* ... */). The comments line begins with a maximum length comment, marked by "MaxLen:". The maximum length value is given in bytes and reflects the real estate constraints for this particular string for a character display.

For example, the maximum length for test\_p\_1 above is 21. Although the English string "New Cust Address" is only 16 bytes long, a maximum length of 21 means that translated strings can be (but need not be) 5 bytes longer than the English string.

Comment

Comment is the description of string that will help to clarify the meaning or context of the string. This comment should come directly after the word "Comment:"

Give full text of all translatable abbreviations

Whenever an abbreviation or acronym is used in the translatable text, a comment should be entered that gives the full version of the abbreviated term.

Rules for creating/modifying any translatable Hard-coded string:

1. DO NOT build translatable strings from more than one string. Such individual strings are not translatable. (see standard STD-0026)

2. DO NOT use special characters like "~" as part of translatable strings. Example:

   \[
   \text{&SCOPED-DEFINE test\_p\_1 } \text{"One~nTwo"}
   \text{message } \{\text{&test\_p\_1}\} \text{view-as alert-box.}
   \]

   This code doesn’t compile, so split the string into coherent, stand-alone and meaningful strings. The above code can be written as:

   \[
   \text{&SCOPED-DEFINE test\_p\_1 } \text{"One"}
   \text{&SCOPED-DEFINE test\_p\_2 } \text{"Two"}
   \text{message } \{\text{&test\_p\_1}\} + \text{"~n"} : \text{U} + \{\text{&test\_p\_2}\} \text{view-as alert-box.}
   \]

3. Messages should not be hard-coded in the source code, but should reside in msg_mstr. (see standard STD-0071)

4. Avoid ambiguous hard-coded strings; be clear and concise in English. (see standard STD-0170)

5. English is an extremely concise language, so allow enough space for each label to be translated to a longer word. (see standard STD-0024)

6. Be consistent. Make any abbreviated labels consistent with other MFG/PRO abbreviations. (see standard STD-0037)

Right

Code
QAD Development Standards

```plaintext
/* ********** Begin Translatable Strings Definitions ********** */
&SCOPED-DEFINE test_p_1 "New Cust Address"
   /* MaxLen:21 Comment: New Customer Address. */
&SCOPED-DEFINE test_p_2 "Old Cust Address"
   /* MaxLen:21 Comment: Old Customer Address. */
/* ********** End Translatable Strings Definitions ********** */

define new shared variable addr2 like ad_addr label {&adadcg_p_1}.
define new shared variable addr1 like ad_addr label {&adadcg_p_3}.
define variable name1 like ad_name no-undo.
define variable name2 like ad_name no-undo.

form
  addr1 colon 22
  name1 no-label
  addr2 colon 22
  name2 no-label
  with frame a side-labels width 80 attr-space.

Wrong

Code

```plaintext
/* ********** Begin Translatable Strings Definitions ********** */
&SCOPED-DEFINE test_p_1 "New Cust Address"
   /* MaxLen:16 Comment: */
&SCOPED-DEFINE test_p_2 "Old Cust Address"
   /* MaxLen:16 Comment: */
/* ********** End Translatable Strings Definitions ********** */

define new shared variable addr2 like ad_addr label {&adadcg_p_1}.
define new shared variable addr1 like ad_addr label {&adadcg_p_3}.
define variable name1 like ad_name no-undo.
define variable name2 like ad_name no-undo.

form
  addr1 colon 22
  name1 no-label
  addr2 colon 22
  name2 no-label
  with frame a side-labels width 80 attr-space.

See Also
N/A
STD-0046 Do not use the meta schema to get information from the data dictionary

Summary

Do not use the meta-schema to get information from the data dictionary which needs to be available in the native language during program execution.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0046</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Translatable Strings, DB Tables, DB Fields</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Do not use the meta-schema to get information which needs to be available in (translated to) the native language during program execution. The programs are compiled against native language databases but may be run against a database that uses a different language. The meta-schema information may not be translated. If the information is displayed from meta-schema which is not in the user's native language, it may not make any sense to the users.

Examples

Right

Label for Field "pt_part" is "Item Number", so the label retrieved by function getTermLabel() is "Item Number". This label value will be translated into a foreign language even if it is running against a database which does not use that language.

Code

```display getTermLabel("ITEM_NUMBER", 18) format "x(18)" no-label.
```

Wrong

Code

```find _field where _field-name = "pt_part".
if available(_field) then display _field._label format "x(18)" no-label.
```

See Also

STD-0146 File and Field attributes from meta-schema
STD-0047 Message Numbers Reserved for customers and distributors

Summary
Message Numbers in ranges 9000 to 9999 will be reserved exclusively for our customers and distributors

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0047</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Devmstr</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
Message numbers for the msg_det file in the 9000 to 9999 range are reserved exclusively for the use of our customers and distributors.

Incorporation of customer specific or distributor developed functionality into standard product should also include a change to any message numbers that use the described 'reserved' range of message numbers. Once this code becomes standard product, these message numbers should not be referenced.

See References section for link to the mechanism to obtain a new message number.

In order to avoid conflicts in message numbering, it is important to give customers and distributors sequences of message numbers to use.

See Also
STD-0048 Assigning Message Numbers
STD-0048 Assigning Message Numbers

Summary

Use “36.7.7 - R&D Message Maintenance” (devmsgmt.p) to generate the next message number for you by leaving the Message Number field blank.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0048</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Devmstr</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Use “36.7.7 - R&D Message Maintenance” (devmsgmt.p) to generate the next message number for you by leaving the Message Number field blank. Then use the Type field to classify the message. Note: Do not reserve ranges of message numbers for specific applications.

Before creating new messages, use the Unix tool ‘msgck’ to search existing messages to see if one can be used for your intended purpose. Duplicate messages in devmstr should be avoided, and the tool will allow you to search for a keyword or phrase that may be contained in an existing message. Run ‘qhelp msgck’ from your Unix prompt for more info on the search tool.

Use of devmsgmt.p manages the allocation of message numbers so there are no conflicts. Going around the tool may seem like a good idea, for example, to reserve message numbers but it generally fails for two reasons. One is you didn't reserve enough to start with so you have to create another range. Second with all these ranges already reserved, also it's difficult to find a new range for a new application. So allowing the tool to manage the message is much easier and cleaner.
STD-0049 Field Validation

Summary

Any new field validations should be implemented in procedural code rather than via schema validations.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0049</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Procedure Structure, DB Fields</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Effective with this standard, no further schema validations (xxx.v code) should be added to the data dictionary. Rather, required validation logic should be implemented in procedural code. Furthermore, this procedural code should be separated from UI logic and be constructed as ‘internal procedures’ either in existing ‘Responsibility Owning Procedures’ (ROPs) if available or in new external procedures. It is acceptable to make calls to pxmsg.i in a validation procedure. In accordance with Standards prior to Release eB of MFG/PRO, the preferred approach to field validation was to utilize 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 new 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. In addition, the implementation of validation logic in procedural code accomplishes the desirable separation of business logic from the UI and the database. Existing examples of duplicated functionality provide useful reference cases on how to properly construct new field validation procedures. These include the following:

<table>
<thead>
<tr>
<th>ROP Procedure</th>
<th>Schema validation</th>
<th>Internal Procedure</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpcodev.p</td>
<td>validateGeneralizedCodes</td>
<td>gpcode.v</td>
<td></td>
</tr>
<tr>
<td>gpplang.v.p</td>
<td>validateLanguageCode</td>
<td>gpplang.v</td>
<td></td>
</tr>
<tr>
<td>gpsecdp.p</td>
<td>validateFieldAccess</td>
<td>gppswd.v</td>
<td></td>
</tr>
</tbody>
</table>

Write procedural code to call the following internal procedure to validate generalized codes instead of relying on schema validation.

Generalized Code Validation as Implemented in procedural code.
PROCEDURE validateGeneralizedCodes:

Purpose: Standard validation for generalized codes
Exceptions: APP-ERROR-RESULT
Notes:
History:

define input parameter pFieldName1 as character no-undo.
define input parameter pFieldName2 as character no-undo.
define input parameter pFieldValue as character no-undo.
define input parameter pOverrideMsg as character no-undo.

do on error undo, return error {&GENERAL-APP-EXCEPT}:

if (pFieldName2 = "" and not (not can-find(first code_mstr
where code_fldname = pFieldName1
or can-find(code_mstr where code_fldname = pFieldName1
and code_value = pFieldValue)))
or
(pFieldName2 &lt;&gt; "" and not (not can-find(first code_mstr
where code_fldname = pFieldName1
or can-find(code_mstr where code_fldname = pFieldName1
and code_value = pFieldValue))))
then do:

if not msgOverriden(input pOverrideMsg, input pFieldName1) then do:

/* VALUE MUST EXIST IN GENERALIZED CODES */
{pxmsg.i &amp;MSGNUM =716
 &amp;ERRORLEVEL={&amp;APP-ERROR-RESULT}}
end. /* IF NOT MSGOVERRIDEN */
return error {&amp;APP-ERROR-RESULT}.
end. /* IF (PFIELDNAME2 = ""
.... */

end. /* DO ON ERROR */

return {&amp;SUCCESS-RESULT}.
END PROCEDURE.

Generalized Code Validation as Implemented in a schema trigger.

/* gpcode.v - VALIDATE GENERALIZED CODES */
/* (1) "field-name"
/* (2) validation field (defaults to (1)) */

/* The current field is valid if no general codes exist for */
/* the validation field or if it matches one of the codes */
/* defined for this field. */

( "(2)" = "" and (not can-find (first code_mstr where code_fldname = "{1}"
or can-find (code_mstr where code_fldname = "{1}
and code_value = {1}))) )
or
( "(2)" &lt;&gt; "" and (not can-find (first code_mstr where code_fldname = "{2}"
or can-find (code_mstr where code_fldname = "{2}
and code_value = {1}))) )
See Also

N/A
STD-0050 Required Table Fields

Summary

**MFGPRO:**

All new MFG/PRO (qaddb operational) tables must have as a minimum the following fields in the table:

- xx_mod_userid
- xx_mod_date
- xx_user1
- xx_user2
- xx__qadc01
- xx__qadc02

In eB3 and above, all new tables must also have an oid_<table_name> field.

Where:

- xx is the prefix from the table name (e.g. xx = gl where table name = gl_mstr)
- _user represents a field reserved for customer use only
- __qad represents a field reserved for QAD use for temporary schema expansion
- c represents a character field
- <table_name> is the Progress table name (e.g. gl_mstr).

**QAD Financials:**

All new QAD Financials tables must have as a minimum the following fields in the table:

- <tablename>_ID
- LastModifiedDate
- LastModifiedTime
- LastModifiedUser

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0050</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>4</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Tables, DB Fields</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

**Known Exceptions**

QAD Financials "CustomXXX" tables are excluded from this standard. Finance tables have customer/user fields in format "CustomXXX" which are optional, though they are frequently included in the tables.

**Description**

**MFG/PRO Required Fields:**

All new tables must have the following fields as a minimum: xx_mod_userid, xx_mod_date, xx_user1, xx_user2, xx__qadc01, and xx__qadc02. In eB3 and above, all new tables must also have an oid_<table_name> field.

Tables should always contain these required fields so we can test for them in the schema validation scripts and report errors if they don't exist. The table below contains the field details that should be used.

Where xx = table prefix
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Datatype and Format</th>
<th>Label</th>
<th>Column Label</th>
<th>Initial</th>
<th>MAX-WIDTH</th>
<th>SQL-WIDTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>oid_&lt;table_name&gt;</td>
<td>Decimal</td>
<td>?</td>
<td>?</td>
<td>&quot;0&quot;</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>xx_mod_userid</td>
<td>Character &quot;x(8)&quot;</td>
<td>User ID</td>
<td>?</td>
<td>***</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>xx_mod_date</td>
<td>Date &quot;99/99/99&quot;</td>
<td>Modified Date</td>
<td>Mod Date</td>
<td>&quot;today&quot;</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>xx_user1</td>
<td>Character &quot;x(24)&quot;</td>
<td>User Field Character 1</td>
<td>?</td>
<td>***</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>xx_user2</td>
<td>Character &quot;x(24)&quot;</td>
<td>User Field Character 2</td>
<td>?</td>
<td>***</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>xx__qadc01</td>
<td>Character &quot;x(24)&quot;</td>
<td>?</td>
<td>?</td>
<td>***</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>xx__qadc02</td>
<td>Character &quot;x(24)&quot;</td>
<td>?</td>
<td>?</td>
<td>***</td>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>

Note: If QAD-reserved fields are used in code, they should be moved into "real" fields when the next opportunity for schema changes is possible. This is a time-consuming activity. The use of QAD-reserved fields should be done
with great caution as is highlighted in standards STD-0305 and STD-0307 ("See Also" section, below).

If the opportunity arises, projects teams may add the remaining required fields to existing legacy tables, given there’s time for such activity in the project schedule. If this is done, the xx_mod_date field should be added as non-mandatory so that existing customer data will not produce errors when Progress inserts the new field. If the field must be created as mandatory, logic must be added to the conversion suite for that release to set a valid date in the new field for our customers' data.

The oid_<table_name> field provides a unique record identifier that can be used as a single-field link to audit trails, e-signatures and as a foreign key in the new design of related tables.

The _user fields provide the users with fields with which to extend their use of an existing table without having to modify the table.

The __qad fields can be used for temporary table expansion or for bug fixes.

**QAD Financials Required Fields:**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Datatype and Format</th>
<th>Label</th>
<th>Column Label</th>
<th>Initial</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;tablename&gt;_ID</td>
<td>Integer &quot;9999999999&quot;</td>
<td>ID Field</td>
<td>?</td>
<td>0</td>
<td>Table’s primary index field</td>
<td></td>
</tr>
<tr>
<td>LastModifiedDate</td>
<td>Date &quot;99/99/9999&quot;</td>
<td>&quot;Last Modified Date&quot;</td>
<td>&quot;Mod Date&quot;</td>
<td>?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LastModifiedTime</td>
<td>Integer &quot;99999&quot;</td>
<td>&quot;Last Modified Time&quot;</td>
<td>&quot;Mod Time&quot;</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LastModifiedUser</td>
<td>Character &quot;x(20)&quot;</td>
<td>&quot;Last Modified User&quot;</td>
<td>&quot;Mod User&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example**

Assume that usrg_mstr is a new table being added to the MFG/PRO qaddb schema in eB3.

**Right**

**Code**
ADD TABLE "usrng_mstr"

AREA "Schema Area"
LABEL "User Group Master"
DESCRIPTION "User Group Master"
DUMP-NAME "usrng_mstr"
TABLE-TRIGGER "Create" NO-OVERRIDE PROCEDURE "usr gc t" CRC \\

ADD FIELD "old_usrng_mstr" OF "crm_mstr" AS decimal
DESCRIPTION "Unique application generated record identifier."
FORMAT ">>>>>>>>>>>>>>>>>>>>>>>>>>9.9<<<<<<<<<"
INITIAL "0"
MAX-WIDTH 38
DECIMALS 10
MANDATORY

ADD FIELD "usrng_mod_date" OF "crm_mstr" AS date
DESCRIPTION "The date of last change"
FORMAT "99/99/99"
INITIAL "today"
LABEL "Modified Date"
COLUMN-LABEL "Mod Date"
MANDATORY

ADD FIELD "usrng_mod_userid" OF "crm_mstr" AS character
DESCRIPTION "The user ID of the user who made the last change"
FORMAT "x(8)"
INITIAL ""
LABEL "User ID"
MANDATORY
MAX-WIDTH 80

ADD FIELD "usrng_user1" OF "crm_mstr" AS character
DESCRIPTION "Customer reserved field"
FORMAT "x(24)"
INITIAL ""
LABEL "User Field Character 1"
MANDATORY
MAX-WIDTH 80

ADD FIELD "usrng_user2" OF "crm_mstr" AS character
DESCRIPTION "Customer reserved field"
FORMAT "x(24)"
INITIAL ""
LABEL "User Field Character 2"
MANDATORY
MAX-WIDTH 80

ADD FIELD "usrng__qadc01" OF "crm_mstr" AS character
DESCRIPTION "QAD reserved field"
FORMAT "x(24)"
INITIAL ""
MANDATORY
MAX-WIDTH 80

ADD FIELD "usrng__qadc02" OF "crm_mstr" AS character
DESCRIPTION "QAD reserved field"
FORMAT "x(24)"
INITIAL ""
MANDATORY
MAX-WIDTH 80
See Also

STD-0006 Allowed number of fields per table
STD-0061 Make all fields 'Mandatory'
STD-0051 Providing for Extra Customer Fields
STD-0305 Extending frozen schema tables to provide extra QAD fields (eB2 and up)
STD-0307 Providing for Extra QAD Fields
STD-0326 Every qaddb table must have a create trigger (eB3 and up)
STD-0333 Database fields with Extents are not supported
STD-0051 Providing for Extra Customer Fields

Summary

Guideline for providing extra customer (user) fields to new and existing tables, including their allowed schema attributes.

<table>
<thead>
<tr>
<th>ID</th>
<th>Version</th>
<th>Language</th>
<th>Status</th>
<th>Categories</th>
<th>Applicability</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD-0051</td>
<td>4</td>
<td>Progress</td>
<td>Published</td>
<td>DB Fields</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

In an effort to make the enhancement of MFG/PRO and other QAD products easier for our end customers, extra customer (also called user) fields should be provided in an application's database schema. In standard STD-0050: Required Table Fields ("See Also" section, below) there are a number of fields identified as mandatory (particularly, the two character datatype user fields named _user1 and _user2). If extra fields are desired in a table due to a high likelihood of the need for customization, additional customer fields may be added according to the allowed schema attributes list contained in this document.

Guideline for adding additional customer fields:

Each additional _user field will contain a datatype indicator (such as: c, d, i, l, or t) and a numeric value (for example: 03, 04, ...). The reason for this is to allow for more _user fields than the required fields.

If the table has little growth potential, then the mandatory fields are sufficient. For example, the Alternate Unit of Measure Master Table (um_mstr) is considered a low-growth table and so has only the required user fields _user1 and _user2. However, if the table has an average growth potential, it might be a good idea to have 5 user character datatype fields and 2 fields of each other datatype: decimal, integer, logical and date. Control files would be good examples of tables with average growth potential. If the table has large growth potential, it might be a good idea to have 10 user character fields and 3 fields of each other datatype (decimal, integer, logical and date). For example the Customer Master Table (cm_mstr) has large growth potential.

This is strictly a judgment call based on the understanding of the use of the table. Creation of extra user fields of any datatype is left to the developer's discretion; those fields can be created or not depending on if the development group thinks they might be needed. The goal is to provide for probable future end-user customization, while trying to minimize the creation of excessive fields that will not be used.

Schema Attributes:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Datatype &amp; Format</th>
<th>MAX-WIDTH</th>
<th>Label</th>
<th>Column Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx_userc03</td>
<td>Character x(24)</td>
<td>80</td>
<td>User Field Character 3</td>
<td>?</td>
<td>Customer reserved field. Note, the numbering has started at 03 because two user character fields xx_user1 and xx_user2 are mandatory.</td>
</tr>
<tr>
<td>xx_userd01</td>
<td>Decimal &gt;&gt;&gt;&gt;&gt;&gt;9.9&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;</td>
<td>38</td>
<td>User Field Decimal 1</td>
<td>?</td>
<td>Customer reserved field</td>
</tr>
<tr>
<td>xx_useri01</td>
<td>Integer &gt;&gt;&gt;&gt;&gt;&gt;9</td>
<td>n/a</td>
<td>User Field Integer 1</td>
<td>?</td>
<td>Customer reserved field</td>
</tr>
</tbody>
</table>
QAD Development Standards

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
<th>History</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx_userl01</td>
<td>Logical</td>
<td>yes/no</td>
<td>Where: xx is the prefix from the table name. (For example: xx = gl where table name = gl_mstr)</td>
</tr>
<tr>
<td>xx_userl01</td>
<td>Date</td>
<td>99/99/99</td>
<td>xx_userl01 is the prefix from the table name. (For example: xx = gl where table name = gl_mstr)</td>
</tr>
</tbody>
</table>

**Rationale**

Customer fields are for our customers to use. By providing these fields as part of the database we make it easier for our customers to enhance the product.

**Examples**

**Right**

Code
QAD Development Standards

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Table Name</th>
<th>Type</th>
<th>Description</th>
<th>Format</th>
<th>Initial</th>
<th>Label</th>
<th>Mandatory</th>
<th>Max-Width</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>crm_userc03</code></td>
<td><code>crm_mstr</code></td>
<td>character</td>
<td>Customer reserved field</td>
<td><code>X(24)</code></td>
<td>&quot;&quot;</td>
<td>User Field Character 3</td>
<td>MANDATORY</td>
<td>80</td>
</tr>
<tr>
<td><code>crm_userd01</code></td>
<td><code>crm_mstr</code></td>
<td>decimal</td>
<td>Customer reserved field</td>
<td><code>-&gt;&gt;&gt;&gt;,&gt;&gt;&gt;9.9&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;</code></td>
<td>0</td>
<td>User Field Decimal 1</td>
<td>MANDATORY</td>
<td>38</td>
</tr>
<tr>
<td><code>crm_useri01</code></td>
<td><code>crm_mstr</code></td>
<td>integer</td>
<td>Customer reserved field</td>
<td><code>-&gt;&gt;&gt;&gt;&gt;&gt;&gt;9</code></td>
<td>0</td>
<td>User Field Integer 1</td>
<td>MANDATORY</td>
<td>0</td>
</tr>
<tr>
<td><code>crm_userl01</code></td>
<td><code>crm_mstr</code></td>
<td>logical</td>
<td>Customer reserved field</td>
<td><code>yes/no</code></td>
<td>&quot;no&quot;</td>
<td>User Field Logical 1</td>
<td>MANDATORY</td>
<td>0</td>
</tr>
<tr>
<td><code>crm_usert01</code></td>
<td><code>crm_mstr</code></td>
<td>date</td>
<td>Customer reserved field</td>
<td><code>99/99/99</code></td>
<td>?</td>
<td>User Field Date 1</td>
<td>MANDATORY</td>
<td>0</td>
</tr>
</tbody>
</table>

See Also

STD-0006 Allowed number of fields per table
STD-0050 Required Table Fields
STD-0041 Use of user fields in MFG-PRO is NOT allowed
STD-0305 Extending frozen schema tables to provide extra QAD fields (eB2 and up)
STD-0307 Providing for Extra QAD Fields
STD-0333 Database fields with Extents are not supported
STD-0052 CIM Interface Compatibility

Summary
The Progress user interface supports certain features that are not present when running in CIM mode and this should be considered when programming.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0052</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style, APIs, Character UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>All CIM-enabled applications</td>
</tr>
</tbody>
</table>

Description
Most MFG/PRO menu programs can be run in CIM mode. CIM mode imposes certain requirements in addition to those present when running with the user interface. When writing a program that will/could be run by CIM, then these must be taken into account when writing terminal I/O dependent code. Specifically, any logic requiring user interaction not supported by CIM functions (tab, period, etc.), such as pressing the spacebar, must be avoided. This can be accomplished by conditionalizing code execution subject to the setting of the batchrun variable.

Use "if not batchrun" to bracket these instances of Progress code:

- Next-prompt with undo, retry and commands 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 (e.g., the swindow*.i include files).

Other considerations for CIM are:

- PROGRESS does not permit the F5 key to be emulated by CIM, therefore CIM cannot be used to process standard MFG/PRO delete transactions.
- Database validation (e.g., Generalized Codes, Site Security, etc.) does not work 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 MFG/PRO database.
- CIM does function in the NT environment when run in character mode in pre-8.2 Progress versions, but only in file rather than continuous mode because Progress did not support pipes on NT until 8.2. Even on Progress 8.2 and after, large CIM transactions can cause errors because NT support for pipes is very limited and more than about 100 characters in a single transaction can exceed the NT size limits. CIM works in the NT environment for MFG/PRO character only, but support for continuous processing is non-existent before Progress 8.2 and limited in higher versions of Progress.
- Code should be sensitive to the need for Endkey and accommodate the period (which emulates the Endkey) when Endkey processing is applicable in the UI to advance to the next frame.
- CIM files should contain three Endkey emulations (the period character, each occurring on separate lines) at the end of each CIM transaction. This insures control is returned to the MFG/PRO main menu, regardless of errors/warnings that may have been raised during transaction processing. If this is not done, then any error in the middle of a CIM transaction can cause spurious errors in subsequent transactions, as they may not start their keystroke emulations from the menu screen assumed.
- The use of dynamic pop-up windows inside MFG/PRO cannot be accommodated by CIM when they are displayed only conditionally based on MFG/PRO data values, such as a Control File setting (example: multi-currency pop-ups requesting exchange rates). This factor should be taken into consideration when deciding whether to use CIM in applications with dynamic pop-up windows.
- CIM can have very slow performance due to the overhead of keystroke emulation and parsing as handled by Progress. It should be recommended for use only with fairly simple and/or low-volume transactions.
- Q/LinQ can replace CIM and improve on many of its built-in limitations in many respects, such as runtime performance and the quality of audit trails.

See the links in the Reference section below for further information.
All the information in this guideline is applicable to the Q/LinQ version of CIM, referred to as "UI Emulation" on the Q/LinQ registration screens.

**Rationale**

Certain constructs require a user interface and/or user action to function properly and thus may not operate as desired when run under CIM.

**Examples**

**Right**

**Code**

```c
/* example 1: */
{gprun.i "gpsirvr.p"
   "(input site, input site1, output return_int)"

   if return_int = 0 then do:
     if not batchrun then do:
       next-prompt site with frame a.
       undo mainloop, retry mainloop.
     end.
     else do:
       undo mainloop, leave mainloop.
     end.
   end.

/* example 2: */
if not batchrun then do:
   {swindowa.i
     file=qad_wkfl
     framerate="k"
     record-id=qad_recno
     search=qad_key1
     equality=qad_wkfl_id
     scroll-field=qad_charfld[1]
     update-leave=yes
     display1=qad_charfld[1]
     display2=qad_charfld[2]
     display3=qad_decfld[5] @ open_ref
     display4=qad_charfld[6] @ wo-um
     display5=qad_decfld[6] @ receipt
     display6=qad_charfld[7] @ trans-um1
     display7=qad_decfld[4] @ reject
     display8=qad_charfld[8] @ trans-um2 }

   if keyfunction(lastkey) = "end-error"
   and not lotitm then leave.
   if keyfunction(lastkey) = "end-error"
   and lotitm then do:
     {mfmsg.i 1119 2}
     pause.
   end.
```

**Wrong**

/* example 1: */
{gprun.i "gpsirvr.p"
   "(input site, input site1, output return_int)"

   if return_int = 0 then do:
     if not batchrun then do:
       next-prompt site with frame a.
       undo mainloop, retry mainloop.
     end.
     else do:
       undo mainloop, leave mainloop.
     end.
   end.

/* example 2: */
```c
if not batchrun then do:
   {swindowa.i
     file=qad_wkfl
     framerate="k"
     record-id=qad_recno
     search=qad_key1
     equality=qad_wkfl_id
     scroll-field=qad_charfld[1]
     update-leave=yes
     display1=qad_charfld[1]
     display2=qad_charfld[2]
     display3=qad_decfld[5] @ open_ref
     display4=qad_charfld[6] @ wo-um
     display5=qad_decfld[6] @ receipt
     display6=qad_charfld[7] @ trans-um1
     display7=qad_decfld[4] @ reject
     display8=qad_charfld[8] @ trans-um2 }
```
See Also

STD-0040 Ensure that a user can respond to 'Press space bar to continue' with either the spacebar or end-key (F4-esc)
STD-0080 Proper Use of KEYCODE, KEYFUNCTION and LASTKEY functions
STD-0226 Messages - Bells and Pauses
STD-0057 Deleting Fields

Summary
Do not drop/delete/remove fields from database schema. Obsolete them.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0057</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
Do not drop/delete fields; Obsolete them instead. Dropping/deleting fields causes the table's CRC value to change. We allow this, but we don't allow the number or order of the fields to change because that would cause problems with dumping data from one version of a QAD product and trying to load it into another version where one or more fields were removed.

See Also
STD-0042 Obsoleting Fields
STD-0058 Field Order Number Sequence

Summary

- When adding new fields to a table, make sure the ORDER attribute number is 10 greater than the highest previous field order.
- New order numbers must not skip over a multiple of 10.
- Never add a new field with an order number that is between two existing fields.
- Do not change the order number of any field that is in a released product's schema.
- Do not submit .df files containing ORDER and POSITION attributes for new fields being added to a database.

- Do not write code which depends on a field's specific ORDER or POSITION values.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0058</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

This is true for 92b only. (QAD EE files generated by Build Framework functions automatically include proper ORDER and POSITION values)

Description

Field ORDER attribute values should be multiples of 10. New order numbers must not skip over a multiple of 10. Never add a new field to a table with an order number that is between two existing fields (don't add fields in the middle of a table). New fields added to a table in a development database must have all their order numbers be higher than the highest order number already used in that table (new fields must be added to the end of a table).

When submitting schema changes to the DBA team for a given QAD product, remove all field ORDER and POSITION attributes from the .df file.

This is true for 92b only. (QAD EE files generated by Build Framework functions automatically include proper ORDER and POSITION values)

- Since more than one project team may add a new field to the same table, those project teams may in fact choose to use the same ORDER and POSITION numbers in their development databases. Fields sharing the same ORDER and POSITION will cause errors when the new fields are added to the public baseline databases. When the ORDER and POSITION attributes are not specified in a .df file, Progress will correctly add the new fields with an ORDER 10 greater than the last field (the one with the previously highest ORDER value) on the table and a POSITION 1 greater than the field with the previously highest POSITION value.
- This same rule applies both to fields being added to existing tables as well as to fields of newly added tables. For new tables, as with existing ones, Progress correctly sequences the ORDER and POSITION values of fields in the order in which the fields are listed in the .df file when the .df file does not contain ORDER and POSITION attributes.

Do not change the order number of any field in a released product.

- ORDER numbers affect the sequence in which field data is dumped and loaded. Changes in ORDER sequence can cause dump and load programs to break and adversely affect product upgrade and conversion processes.

Never write code or create application logic which depends on a field's specific ORDER or POSITION values.

- These values will likely change when the schema is promoted to the public baseline development databases, causing the application logic to break or yield unexpected results.
Rationale

ORDER numbers affect the sequence in which data is dumped and loaded. This is how we assure that dump/loads will work correctly and how we check for field name changes.

See Also

STD-0042 Obsoleting Fields
STD-0057 Deleting Fields
STD-0059 RECID data type

Summary

Do not store RECIDs persistently as a foreign key to another record

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0059</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Use sequences instead of RECID if you need to have a unique identifier for records that otherwise are not unique (such as tr_hist and exru_usage).

Do not use a stored RECID as a foreign key to another record.

- 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.

Our dump/load procedures now use the Progress Import and Export verbs which ignore any RECID type fields.

Dump/loads will change the RECID value and invalidate the foreign key relationship. Also, when records are deleted, Progress will reuse the RECID for new records.

Examples

Right

Code

```pascal
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

Code

```pascal
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 re-used, causing duplicate key error */
```

See Also
STD-0011 Maintaining Sequences
STD-0183 Sequence Naming
STD-0060 Help field in field definition

Summary
Leave the "Help" field blank.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0060</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
When defining a field, always leave the "Help" field blank. Any value in the help field will conflict with our display of function key usage.

Rationale
This standard is appropriate for all QAD products, including the FIN schema. Any text stored in the HELP attribute displays in the status area of the application window and would overwrite our standard function key display. So the HELP attribute should remain empty/unused, at least for QAD SE and EE and all bolt-on products which integrate with QAD SE and EE.
STD-0061 Make all fields 'Mandatory'

Summary

When creating new fields in the schema, make them "MANDATORY". Date fields, large objects and raw fields, however, may require special attention.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0061</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Known Exceptions

- This standard applies only to the addition of new fields to the schema. Existing fields should not be modified from non-mandatory to mandatory.
- This standard applies only to non-CBF controlled tables.

Description

When creating new fields, either in new tables or by adding fields to existing tables, specify the Progress MANDATORY attribute to indicate the field requires a value unless there is a specific and valid reason why the unknown value must be supported in the field.

The use of the "Mandatory" flag does NOT mean that the user must enter something other than blank in the field. You can enter blank even if MANDATORY is set to yes. When fields are made mandatory they just cannot contain the unknown value, which is represented by a question mark (?)

- The Progress MANDATORY field attribute gets turned into the NOT NULL constraint on Oracle.
- In Oracle there is no such thing as "the unknown value"; the closest Oracle equivalent is the null value, indicating a nonexistent value. Progress also distinguishes a 0-length string from the unknown value; Oracle does not. Therefore, the DataServer for Oracle maps Progress unknown values (?) to Oracle null values, while 0-length character strings are mapped to a single space (" ") on Oracle.

Application code (not VALEXP schema validations) must be used to check for unknown values regardless of the field's mandatory setting. If a mandatory field can be entered by the user via an input field, the program should explicitly check for the unknown value (?) and issue an appropriate error message. If this check is not made, Progress will issue an error message when an unknown value is entered into a mandatory field.

Character Fields

When character fields are made mandatory, they can still be left blank; they just cannot contain the unknown value.

Numeric Fields

When decimal, integer or int64 fields are defined as mandatory, these numeric fields can still be left zero; they just cannot contain the unknown value. In MFG/PRO we generally don’t allow numeric fields to contain "no number" so they must be assigned default values, usually '0' (zero).

Logical Fields

In MFG/PRO we generally don’t allow logical fields to contain "no choice", so they must be assigned default values, usually 'no'.

Date and Time Fields

Date and time fields (datatypes: date, datetime, etc.) are somewhat problematic, since to display a blank in the
field, the value must be unknown. If made mandatory, however, date and time fields will have to contain a valid
date value which will be displayed. A blank is not a valid date or time.

- Mandatory date fields must specify an appropriate default \texttt{INITIAL} date value, such as: \texttt{INITIAL}
  
  \texttt{today}. This will ensure new records are populated with a valid date value when created, but it does not
  populate the table's existing records.

- Previously existing records in a table in which an existing date field is made mandatory or a new
  mandatory date field is added will still lack valid date values (Progress defaults the field to the
  unknown value in all existing records). Therefore, previously non-mandatory date fields which
  were modified to be mandatory, as well as new mandatory date fields added to an existing table,
  require the developer to create an upgrade/conversion program to populate all existing records in
  the table with a valid date value in this field.

For these reasons, it is recommended that previously existing date fields NOT
be changed from non-mandatory to mandatory and that new mandatory date
fields NOT be added to existing tables in the schema. If you must update a date
field or add a new date field to an existing table as mandatory to ensure it
always has a valid value, you must supply the conversion program to populate
the field.

- New tables may include new, mandatory date and time fields as long as those fields have a valid
  \texttt{INITIAL} value and the application code ensures the field will always contain a valid date or time.
  Otherwise, do not make new date or time fields mandatory.

Recommendation is to NOT make new date and time fields in new tables
mandatory.

- Non-mandatory date and time fields are allowed to contain the unknown value (\texttt{?}). For example: \texttt{INITIAL}
  \texttt{?}

\textbf{BLOB, CLOB and Raw Fields}

When large objects and raw fields are made mandatory, they can still be left blank; they just cannot contain the
unknown value. However, application logic is simpler when large object and raw fields are allowed to be empty
(unknown/null).

Recommendation is to NOT make large object and raw fields mandatory.

\textbf{Special Considerations for All Datatypes}

- Do NOT modify existing schema by changing the status from non-mandatory to mandatory as it could
  cause customer data loads to fail during product upgrades. Modifying an existing field to mandatory
  causes Progress to default the field to the unknown value in all existing records. This then requires the
  developer to create an upgrade/conversion program to populate all existing records in the table with a valid
  value in this new field during product upgrade processing.

- If the field is an element of a unique index and the field is allowed to have a null value, then you can have
duplicate records. Progress does not consider records to be identical if a record exists where one of the
unique key values is "?" and you create a new record with the identical key values. However, Oracle does
consider such records to be identical and therefore in violation of the unique index constraint. The
programmer must perform a "\texttt{FIND FIRST ...}" to confirm that no other record has that set of key
values. Another way of checking is for the program to handle the "Duplicate Unique Key" error whenever it
occurs.

- When a field that contains an unknown/null value is used to control processing via logical statements, you
can get unpredictable results. Programmers must ensure fields contain valid values before being used in a
logical test condition.

- While schema validation (\texttt{VALEXP}) phrases are only used when going from screen buffer to record buffer,
the \texttt{MANDATORY} phrase is checked even if you assign directly to the record buffer. For example, "\texttt{ad_addr}
  = ?" will cause an ad_mstr record update to fail (ad_addr is a mandatory field in ad_mstr). See "STD-0049
Field Validation" ("See Also" Section, below).

- To prevent 0-length and unknown/null character strings (a blank value) from being used in a field, use the
following \texttt{WHERE} clause expression: \texttt{field <> "" and field <> ?}. While it is possible to test for the
0-length and unknown/null values with the expression of \texttt{field > ""}, the test only works for ASCII. If we
end up having to support EBCDIC, then the test will cause normal printable characters to fail the test as well.

**CBF Schema**
For tables controlled through Component Builder Framework (CBF), if a field is defined as mandatory then code is automatically generated to verify that the field has a value. In other words, it DOES mean that the user must enter a non-blank value in the field. CBF has a different mechanism to ensure that duplicate records cannot be created if an index contains fields that are not defined as mandatory. Therefore, this standard should not be applied to CBF controlled tables.

**Rationale**

It is desirable to have some value in all fields. The MANDATORY attribute assures that the field will not be assigned the problematic unknown/null value. Progress uses the null value in Oracle because it is the closest to their "unknown value". The two are similar, but not actually the same. Oracle gets into difficulty when null values are used in indexes.

This standard arose out of problems caused by past MFG/PRO conversions where fields that were not mandatory in a prior release were made mandatory in the new version. Unlike Progress, Oracle does not allow more than 1 null key/value in a unique index. This standard prevents fields being added to an Oracle index when those fields contain the null value.

- Having fields that were not mandatory become mandatory in a new release caused index creation failures on Oracle when duplicate or several unknown (null) values existed in fields that became part of an index in a new release. When fields were altered to be mandatory/not null during a conversion and the incoming data from the old version contained unknown/null values, the conversion process would fail on Oracle systems.
- Problems also occurred when customers migrated from Progress to Oracle due to unknown/null values in indexes (again Progress allows but Oracle does not). Following a dump from Progress, subsequent loads and index creations on Oracle would fail.

Continued use and enforcement of this standard gives developers one less thing to think about since they won't have to produce special conversion processes such as updating a field to contain a value in all records when the field becomes part of an index. The MANDATORY attribute assures that the field will not be assigned the problematic unknown/null value.

**See Also**

STD-0049 Field Validation
STD-0064 Character Field Format length limit

Summary
Do not use a format length greater than 4000 for a character field.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0064</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>3</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
Do not use a format length greater than 4000 for a character field.

If the field length is greater than 4000 characters, it causes difficulty with the DataServer utility which converts the Progress data definitions into an Oracle or MS SQL Server schema, with Oracle being more restrictive:

- When migrating a Progress OpenEdge database to MS SQL Server, a character field whose format is less than "x(8000)" is defined in MS SQL Server as the VARCHAR datatype, while larger character fields are defined as TEXT.
- When migrating a Progress OpenEdge database to Oracle, a character field whose format is equal to or less than "x(4000)" is defined in Oracle as the VARCHAR2 datatype, while larger character fields are defined as LONG.

QAD does not support either the SQL Server TEXT datatype nor the Oracle LONG datatype. Therefore, to ensure consistent application behavior on all databases we are limited to the smaller of the MS SQL Server VARCHAR datatype maximum size and the Oracle VARCHAR2 datatype maximum size, which is the Oracle VARCHAR2 limit of 4000 characters.

Rationale
If we do not follow this standard, we must manually adjust the fields defined in the schema holder and resulting ORACLE or SQL Server database after the DataServer conversion utility has done its work. Also by not following this standard we run the risk of the application accepting long values on Progress databases but encountering errors such as "A column in this row being inserted or updated is too large (4212)" when using Oracle or SQL Server. By adhering to this standard we ensure that long character values are handled consistently regardless of the underlying database being used.

See Also
STD-0090 Schema field formats should support storage requirements for Oracle
STD-0301 Oracle MAX-WIDTH (SQL-WIDTH) must be specified on new database fields
STD-0065 Include files - Complete statements and expressions

Summary

Statements and/or expressions contained in include files must be complete. They should not be used to start or complete statements in the “including source”.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0065</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Procedure Structure, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Include files must contain only complete statements or expressions. Partial statements or expressions that are either started or completed outside of the include file are not to be used.

Rationale

Placing fragments of statements or expressions in an include file makes it difficult to see the scope or structure of the complete statement. In the best case, this leads to code that is difficult to read. In the worst case, it can lead to programming errors as a result of the unclear syntax that is required in the source file using the include.

Examples

Right

Code

Here "mfdtitle.i" is completely self contained. Note that while \{gpvar2.i\} appears in the middle of a statement, it neither begins or ends it. This is important for keeping visibility to the span of control.

```plaintext
main.p
/* main.p - Demo program */

{mfdtitle.i}

var1 = var1 + \{gpvar2.i\} . &lt;--- Notice the period here is external to the include file
....

/* end of main.p */

gpvar2.i
/* gpvar.i - A general purpose variable to add anywhere */

var2
```

Wrong
Code

```pascal
main.p
/* main.p - Demo program */

{startcmt.i} end of a comment */   \&lt;--- This statement is started inside of the include file

var1 = var1 + {gpvar.i}          \&lt;--- This statement is ended inside of the include file
....

/* end of main.p */

starcmt.i
/* startcmt.i - Start a comment */

/

{gpvar.i}
/* gpvar.i - A general purpose variable to add anywhere */

var2.
```

See Also

N/A
STD-0066 Do not use Packed Fields

Summary

Do not use a packed field.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0066</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields, Field Access &amp; Storage, Data Handling Logic</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Do not pack a field by storing data beyond the output/display format defined in the database. Packing (or concatenating) fields is a technique employed to put what is logically two or three fields or more into a single field. Often this was done in past MFG/PRO releases to avoid modifications to the schema. QAD applications should define field display formats large enough to contain the maximum data length expected in a field. If this is not possible (due to screen real estate availability, for example), the field's MAX-WIDTH value must be made large enough to contain the maximum data length expected.

Handy Hint

Developers and testers can run sessions with the Progress `-checkwidth` client startup parameter to determine if field data exceeds the metaschema `_width` value (MAX-WIDTH attribute). This startup parameter lets you impose the SQL requirement that data cannot exceed the size specified in `_width` / MAX-WIDTH. Valid parameter modes are:

- `-checkwidth 0` Ignore `_width` value and store the data regardless of size (Default behavior).
- `-checkwidth 1` Stores the data but generates a warning message when data exceeds the `_width` size. Use this mode to detect inadequate MAX-WIDTH values while not interrupting application tests.
- `-checkwidth 2` Does not store the data and generates an error message when data exceeds the `_width` size. Use this mode to force Progress to behave like SQL databases.

Rationale

The use of packed fields is not a standard relational technique. It is not normalized data. The use of packed fields makes it difficult to use standard query tools to retrieve information from the field. It requires some form of parsing program.

When Oracle or SQL Server defines a character field, the size is an upper limit. Progress has no such definition; all Progress character fields can hold 32K strings. The Progress format only defines the output format of the field. Despite the best intentions of programmers, the size of packed data in fields often exceeds the size of the `FORMAT` statement. When the Progress DataServer tries to define the field for Oracle or SQL Server, the Progress utility can use either the field's `FORMAT` or MAX-WIDTH values for the Oracle column width. Actual data lengths which exceed the Progress display format make it impossible to base Oracle or SQL Server column widths on the `FORMAT` value. If the `FORMAT` cannot be large enough to display all expected data in a field, the MAX-WIDTH must be made large enough to hold the expected data.

Examples
**Right**

**Design**

Example 1. The field xx_id is defined with the same or larger format as the values it will store.

Example 2. Several values aren’t packed into one field.

**Code**

Example 1.
Field xx_id is defined in the DB as character, format "x(18)".

```plaintext
assign xx_id = pt_part.
```

Example 2.

```plaintext
assign xx_addr = ad_addr
xx_nbr = so_nbr
xx_part = pt_part.
```

**Wrong**

**Design**

Example 1. A string that can potentially be longer than x(8) is stored in xx_id. This causes a serious error in Oracle if the Oracle schema doesn’t allow for the true size of the values assigned to xx_id.

Example 2. Three fields are packed (concatenated) into one. This is bad design and it makes retrieval of the data difficult. It is also storing a string that is too long for the format of the field (see Example 1).

**Code**

Example 1.
Field xx_id is defined in the DB as character, format "x(8)".

```plaintext
assign xx_id = pt_part.
```

Example 2.

```plaintext
assign xx_id = ad_addr + so_nbr + pt_part.
```

**See Also**

STD-0043 Field Formats Allowed
STD-0064 Character Field Format length limit
STD-0090 Schema field formats should support storage requirements for Oracle
STD-0301 Oracle MAX-WIDTH (SQL-WIDTH) must be specified on new database fields
STD-0069 Use of 'no-error' clause in 'Find' statements

Summary
When performing a "find" statement, use the "no-error" clause and then test for availability. Issue error messages whenever possible.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0069</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Table Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All new or modified code</td>
</tr>
</tbody>
</table>

Description
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's a chance that the record doesn't exist, display an appropriate message and undo, retry, leave, etc. (For Oracle performance reasons, it is acceptable to use the "find" only if the record is accessed with "exclusive-lock", while it is always acceptable to use the "for first" statement).

However it is appropriate to not use the "no-error" clause when the record should be there and the design specifies that default Progress error handling is to be used. This is particularly true when re-finding a record that should already be in the buffer space for a process, such as legacy code which re-finds a record using the recid.

Note: Use of the "for first" statement is preferred over use of a "find" statement. See STD-0164 for additional details.

It is our intent to "capture" as many errors as possible within MFG/PRO so that PROGRESS does not "kick out" the user and lose the transaction being performed due to a PROGRESS error.

Examples
Below, a typical non-exception case is shown.

Right

Code
```plaintext
find pt_mstr exclusive-lock where pt_part = partNumberDesired no-error.
if available pt_mstr then do:
  assign
  pt_um = partUM
  pt_type = partType.
end.
```

Wrong

Code
```plaintext
find pt_mstr exclusive-lock where pt_part = partNumberDesired.
assign
  pt_um = partUM
  pt_type = partType.
```
See Also

STD-0164 Read unique records - 'FIND'
STD-0070 Use four-digit years in hard-coded dates

Summary

When using hard-coded dates, all four digits of the year must be used.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0070</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Logic for Dates</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Avoid the use of hard-coded dates in the code. When it is necessary to use a hard-coded date, all four digits of the year must be used. Using only two digits of the year in a hard-coded date introduces ambiguity into the code and, when displayed, can result in different answers depending on the end user's -yy setting.

Examples

With a -yy setting of 1950, the statement

```
display 01/01/50 format "99/99/9999"
```

results in 01/01/1950. If the -yy setting is changed to 1951 or higher, the same statement results in 01/01/2050. If the programmer only uses 2 digits of the year then it is ambiguous which century is intended. The following code samples assume that you are starting a database with -yy 1970.

```
display 01/01/69 format "99/99/9999".
display 01/01/69 format "99/99/9999".  /* This will result in 01/01/2069, not 01/01/1969 as expected. */
```

See Also

N/A
STD-0071 Location of messages

Summary

Put all messages in msg_mstr

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0071</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Program Style, Translatable Strings</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Messages should not be hard-coded in the source code, but should reside in msg_mstr.

Rationale

Displayable text should not be hard-coded because such strings are difficult to translate.

Examples

N/A

See Also

N/A
STD-0072 Use of DISPLAY and SET Statements vs UPDATE

Summary

Use of the DISPLAY and SET statements is preferable to the UPDATE statement in blocks with UNDO conditions.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0072</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Procedure Structure, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

There is a subtle difference in the behavior between an UPDATE statement and the very similar DISPLAY and SET combination.

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.

This difference is particularly important when error checking follows the editing. If UPDATE is used and a field fails the error check, all fields are returned to their original values, wiping out the value of the field in error as well as all of the entered fields. With SET, the illegal value will remain displayed as will the previously edited values of all the other fields.

Examples

Exceptions:

1. Floating Point fields

   If a floating point field is displayed with less decimal precision than is stored, SET will change the value to the lesser precision even when the field has not been modified. UPDATE will leave the field at full precision. In this case UPDATE is preferable to SET.

   E.g.: schema field x is decimal with format ">>9.999"
   
   X has initial value of 1.115
   
   display x format ">>9.99". - displays the value 1.12 for x
   set x . - sets the value 1.12 for x
   A Go (F1/F2) without modification results in x = 1.12 (bad)

   update x format ">>9.99". - displays and sets the value 1.12 for x
   A Go without modification leaves x = 1.115 (good)

2. Date fields

   For date fields there is also a difference in behavior between the SET and UPDATE statements. This difference occurs when the date to be displayed falls outside the current century window as defined by the Progress -yy startup parameter.
Behavior: when Progress attempts to display a date field with format "99/99/99" and the date value falls outside the century window an error message is generated and the date is displayed as /???. If the UPDATE statement is used for setting the date value and the date on the screen is not modified, the original value of the date is retained.

However, if the SET statement is used for setting the date value and the date on the screen is not modified, the original value of the date is replaced with that shown onscreen, the unknown value (/??). In this case UPDATE is preferable to SET even though the error handling recovery is not as user-friendly.

Note: adding an "if retry then assign" statement before the UPDATE statement to retain the previously entered values so the user does not have to re-enter them does not help this error handling issue. This is because the assign of a date outside the current century window would flag the field as being updated (with a value that could not be processed), a behavior which a simple UPDATE statement would not trigger. If Progress thinks the contents of the input field are changed then the validation fires and the user has to provide a valid input. This is not possible if the desired date falls outside the current century window and only a 2-digit year is allowed for input. Use of only the UPDATE statement (without the assign) would leave an unmodified field unflagged as "modified" and the original value would not be changed if the user did not type into the field.

To prevent values entered on the screen from reverting back to original values when edit validation fails.

```plaintext
/* VARIABLES x, y, AND z RETAIN THE VALUES ENTERED AT */
/* THE set STATEMENT WHEN THE FIELDS FAIL EDITING. */

display x y z with frame a.

editloop:
do on error undo, retry:
set x y z with frame a.
if not {gpvaly.i y} then do:
{mfmsg.i 123 3}
next-prompt y with frame a.
undo editloop, retry.
end.
if not {gpvalz.i z} then do:
{mfmsg.i 234 3}
next-prompt z with frame a.
undo editloop, retry.
end.
end.

/* VARIABLES x, y, AND z ARE RESET TO THEIR ORIGINAL */
/* VALUES WHEN THE FIELDS FAIL EDITING. */

editloop:
do on error undo, retry:
update x y z with frame a.
if not {gpvaly.i y} then do:
{mfmsg.i 123 3}
next-prompt y with frame a.
undo editloop, retry.
end.
if not {gpvalz.i z} then do:
{mfmsg.i 234 3}
next-prompt z with frame a.
undo editloop, retry.
end.
end.
```

See Also

N/A
STD-0073 Frame Titles

Summary

When frame titles are used, words should be shown using title case (the first character of the words should be capitalized.)

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0073</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Translatable Strings</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities (MFG/Pro versions 9.0 and lower)</td>
</tr>
</tbody>
</table>

Description

When frame titles are used, words should be shown using title case (the first character of the words should be capitalized.) This complies with the specifications of the translations team. A space should be included before the first word and after the last word to make the title more readable. Use the "color normal" phrase to prevent the title from appearing in an inverse video block.

Rationale

Enhanced readability and visual cues for user. Simplified work load for translations team.

Examples

Right

Design

Code

```c
/* ********** Begin Translatable Strings Definitions ********** */
&SCOPED-DEFINE adcsmt02_i_4 " Customer Data " /* MaxLen:78 Comment: Frame Title */

/* ********** Begin Translatable Strings Definitions ********** */

form
  cm_taxable  colon 35
  cm_taxc     no-label
  cm_pr_list2 colon 35
  cm_pr_list  colon 35
  cm_fix_pr   colon 35
  cm_class    colon 35
  cm_partial  colon 35
with frame b2 title color normal [&adcsmt02_i_4]
  side-labels width 80 attr-space.
```

Wrong

Design
In following code, frame title has no spaces before and after the title string.

**Code**

```plaintext
/* ********** Begin Translatable Strings Definitions ********** */

&SCOPED-DEFINE adcsmt02_i_4 "Customer Data"
/* MaxLen:78 Comment: Frame Title */

/* ********** Begin Translatable Strings Definitions ********** */

form
    cm_taxable colon 35
    cm_taxc no-label
    cm_pr_list2 colon 35
    cm_pr_list colon 35
    cm_fix_pr colon 35
    cm_class colon 35
    cm_partial colon 35

    with frame b2 title color normal {&adcsmt02_i_4}
    side-labels width 80 attr-space.

See Also

N/A
STD-0075 RECID use for Record Availability in Oracle

Summary

If a newly created database record must be subsequently retrieved from the database before the creation transaction commits, the RECID() function should be used at the time of creation to avoid duplicate record errors and to provide consistent record access in both Progress and Oracle.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0075</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>4</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Table Access &amp; Storage, Procedure Structure</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities involving database record creation</td>
</tr>
</tbody>
</table>

Description

If a newly created database record must be subsequently retrieved from the database before the creation transaction commits, use the RECID() function at the time of creation to avoid duplicate record errors. This typically occurs when the record must be read from the database in a subprogram called from the create program, or when the logic returns to a calling program and needs to read the new record. New database records created in Progress are visible when the index information is written. In Oracle, the records are made visible when the record is committed to the database. This can cause duplicate record errors when committing to Oracle and lack of data visibility when records are read after the create in the same transaction.

These errors can occur because of restrictions Oracle places on record availability. Oracle writes records after the end of a transaction. (Progress writes records to the database as soon as the index information is available.) Therefore, when running on Oracle, newly created records are not available to subsequent database reads within the same transaction, causing them to fail to find the new record or create unnecessary duplicate records.

A call to RECID() forces Oracle to write a new record to the database before the end of the transaction and before any subsequent reads on the record. However, it is not acceptable to execute RECID() after every database record creation, because that only decreases the performance of MFG/PRO on Oracle due to excess database commits. Committing records in Oracle is expensive in terms of performance, and new records should be committed at the end of transactions if at all possible. The use of buffer parameters passed to subprograms or back to calling programs should lessen the need to retrieve a newly created record from the database in the subsequent programs, and hence lessen the need to use the RECID() function.

Since a RECID() call is not guaranteed on every database record creation, it is the responsibility of the developer who retrieves the record later in a subsequent program to make sure the record has been committed to Oracle either through the end of a transaction or a RECID() call.

If the RECID() function is needed, the call should be made AFTER the primary key values are set in the created buffer. If it's used before, you still have the duplicate key or record not available problems. It is vital that the primary index values are set to ensure the record is unique.

It is sufficient to merely access the RECID() function. It is not necessary to save or use the RECID() return value. If it is necessary to save the unique ID of a record, ROWID() is acceptable to use, and will perform a similar write to the database of a new record as RECID(). This standard is designed to ensure compatibility between ORACLE and PROGRESS data visibility in subsequent programs, and RECID() was chosen because of its extensive, and thoroughly-tested, use throughout the current base of MFG/PRO code.

NOTE: The use of RECID() merely to commit a temp-table or work file record should never be done, since these records are only maintained in memory and have no effect on the Oracle database.

To summarize:

- If you are creating a database record, only use RECID() to commit the record if you know the record must be retrieved from the database prior to the end of the transaction.
- If you are reading a database record that might have been created within the same transaction, you must make sure that the creation includes a call to RECID().
Whenever possible, try to pass buffer parameters of newly created database records to subprograms and back to calling programs when needed, rather than having them retrieve the record from the database.

Rationale

This standard insures that newly created records are available for retrieval in subsequent programs within the same transaction for Oracle as well as Progress. Using RECID() avoids one but not all of the causes of duplicate record errors in Oracle, that do not occur when running Progress. However, the use of RECID() to commit records in Oracle prior to the completion of a transaction should be minimized to avoid any adverse performance impact.

Examples

A newly created part is expected to be processed by a subprogram. A subprogram is called to count all parts. The new part will be counted in the right code. The new part will not be counted by the wrong code.

Right

Design

Code

```plaintext
create pt_mstr.
assign pt_part = "99999".
if recid(pt_mstr) = -1 then .
{gprun.i ""countparts.p""
```

Wrong

Design

Code

```plaintext
create pt_mstr.
assign pt_part = "99999".
/* countparts.p will not find the new record under Oracle */
{gprun.i ""countparts.p""
```

See Also

N/A
STD-0078 Translated Database Tables

Summary

Avoid putting translatable text in database tables that will not be translated.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0078</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Translatable Strings, Field Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

The only database tables that are translated are the following:

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg_mstr</td>
<td>Message Master</td>
</tr>
<tr>
<td>lngd_det</td>
<td>Language Detail</td>
</tr>
<tr>
<td>mnt_det</td>
<td>Menu Title Detail</td>
</tr>
<tr>
<td>mnts_det</td>
<td>Menu Substitution Text Details</td>
</tr>
<tr>
<td>lbl_mstr</td>
<td>Label Master (Available for MFG/PRO versions eB and above)</td>
</tr>
<tr>
<td>cd_det</td>
<td>Master Comment (Available for MFG/PRO versions eB and above)</td>
</tr>
</tbody>
</table>

Text will not be translated into foreign language if the text is not stored in translated tables.

See Also

N/A
STD-0080 Proper Use of KEYCODE, KEYFUNCTION and LASTKEY functions

Summary

This document discusses the proper use of the Progress functions KEYCODE, KEYFUNCTION and LASTKEY with respect to keyboard mappings.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0080</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Progress maps certain keystrokes into events or commands sent to the application. Unfortunately, the mappings are different when running a Windows GUI client than they are when running a character mode client. In addition, there are cases where the same command may map to a Ctrl- combination or other keystroke as well as to a function key. The application code must be written in such a way that it will not cause errors which are dependent on which user interface is used.

For example, when the user presses the F1 key in character mode, his intention from a programmatic standpoint is to send the GO command to the application, not to just press the key labeled "F1". Similarly in GUI when the user presses F2, he again is sending the GO command. These two actions, different at the atomic level, both map to the same command to the application, namely "GO". The code should be written so as to remove any UI dependencies wherever possible.

F1-F4 Events:
When checking for what have traditionally been referred to as F1-F4 events, always use syntax similar to the following:

```java
if keyfunction(lastkey) = "value" then ...
```

where value is the command represented by the key, not the label of the key.

I.e., instead of:

F1, you would check for: GO
F2, you would check for: HELP
F3, you would check for: INSERT-MODE
F4, you would check for: END-ERROR

When using keyfunction(lastkey) = "GO" you may also need to test for keyfunction(lastkey) = "RETURN" in the case where "RETURN" would behave like the "GO" function.

Since the CIM processor treats a period (‘.’) on a line by itself as the equivalent of the END-ERROR key (F4/Esc), programs which may run under CIM should also check for this value when END-ERROR is checked.

F5 Event, MFG/PRO Delete Record Action: There is a difference between character and GUI response to the Ctrl-D trap. The delete record action should be coded as follows:

```java
if /*V8-*/
keyfunction(lastkey) = "get"
/*V8+*/
/*V8!
keyfunction(lastkey) = "delete-character" /* GUI response to the
Delete key */
or lastkey = keycode("F5")
or lastkey = keycode("CTRL-D") then ...
```
F6-F12 Events - OK To Use The KEYCODE Function:
The KEYCODE function evaluates a key label (such as F1) for a key in the predefined set of keyboard keys. Since
F6-F12 are trapped in the same manner for character and GUI, it is appropriate to code as follows:

```
if lastkey = keycode("F6") then ...
```

The meanings of these keys are generally dependent on the context in which they are used, and the status area at
the bottom of the screen is used as a prompt to the user. In these cases it is appropriate to evaluate keylabel itself.

**Rationale**

Keyboard mappings via Progress into "standard" functions/events are different in character mode than in GUI. In
general, the application code should not assume one or the other, but should be written so as to accommodate
either.

**Example**

**Right Design**

**Code**

```c
/* These will work as intended whether in character mode or in GUI */

/* 1. */ if keyfunction(lastkey) = "GO"
    or keyfunction(lastkey) = "RETURN"
    then ...  
/* 2. */ if keyfunction(lastkey) = "HELP" then ...
/* 3. */ if keyfunction(lastkey) = "INSERT-MODE" then ...
/* 4. */ if keyfunction(lastkey) = "END-ERROR"
    or keyfunction(lastkey) = "."  
    then ...
/* 5. */ if /* V8- */
    keyfunction(lastkey) = "get"
    /"V8+"/
    /"V8!"
    keyfunction(lastkey) = "delete-character" /
    or lastkey = keycode("F5")
    or lastkey = keycode("CTRL-D") then ...
/* 6. */ if lastkey = keycode("F6") then ...
/* 7. */ if lastkey = keycode("F7") then ...
/* 8. */ if lastkey = keycode("F8") then ...
/* ... etc... */
```

**Wrong Design**

**Code**

```c
/* These might work in character mode but not as intended in GUI */

/* 1. */ if keyfunction(lastkey) = "F1" then ...
/* 2. */ if keyfunction(lastkey) = "F2" then ...
/* 3. */ if keyfunction(lastkey) = "F3" then ...
/* 4. */ if keyfunction(lastkey) = "F4" then ...
```

**See Also**

N/A
STD-0089 low_date and hi_date global variables - value

Summary

MFG/PRO's global shared variable low_date must have the value 01/01/1900, and hi_date must have the value 12/31/3999.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0089</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Logic for Dates</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

MFG/PRO's global shared variable low_date must have the value 01/01/1900, and hi_date must have the value 12/31/3999.

The global shared variables low_date and hi_date represent the minimum and maximum date values that are expected to be stored in the MFG/PRO database. These variables are intended solely for range checking on date fields, and their primary use is in situations where "From" and "To" Dates are used as selection criteria for a report and the user does not enter a specific date range. The input variables are typically set to low_date and hi_date to allow the retrieval of all dates.

low_date and hi_date are defined in mfdeclre.i and are initialized in mf1a.p for character and gui clients and in mfwb02.p for NetUI clients.

Rationale

Oracle has defined their date range to be from 01/01/4712 BC to 12/31/4712 AD. PROGRESS defines their range as 01/01/32767 BC to 12/31/32767 AD. To provide compatibility with both of these databases, the global shared variable low_date should have a value of not less than 01/01/4712 BC and hi_date should have a value not greater than 12/31/4712.

In addition, it is important that the values selected for low_date and hi_date bound the range of conceivable dates that will potentially be stored in the database by the enduser. Thus, the value of hi_date must be set greater than or equal to the highest conceivable date that will be stored in the MFG/PRO database and the value of low_date must be set less than or equal to the lowest conceivable date that will be stored in the MFG/PRO database.

The standard values are noted above.
STD-0090 Schema field formats should support storage requirements for Oracle

Summary

Schema field formats should support storage requirements for Oracle.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0090</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields, Field Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

In PROGRESS, the format of a field defined in the schema is used for display purposes only and Progress allows a field to hold more data than the format will display. In ORACLE the format of a column defines the maximum width for that column (the maximum number of characters that can be stored in the field). To provide an initial database to the end-user, each database column needs to have an appropriate width defined. A width too large will result in performance degradation, while a width too small will result in unexpected query behavior or run-time errors.

Column widths must be large enough to hold the largest data value expected in the field. Specify column widths in Progress via the MAX-WIDTH field attribute (formerly SQL-WIDTH prior to OE10). For detailed instructions, refer to standard STD-0301 in the "See Also" section below.

MAX-WIDTH is used to produce appropriate Oracle column widths when converting Progress databases to Oracle. If an Oracle column definition is not large enough to hold the data, the Progress error "A column in this row being inserted or updated is too large (4212)" occurs and the record write fails.

**temp-table and work-table usage:**

This Oracle storage consideration does not apply explicitly to Progress temp-tables and work-tables. However, a temp-table used to transfer records to an Oracle schema table still encounters the same restriction. In other words, temp-table fields should not be larger than the schema table fields they will populate.

In Progress database fields, MAX-WIDTH also specifies the maximum amount of data that SQL clients such as ODBC products can access. SQL clients and ODBC tools can only access the portion of a data value up to the MAX-WIDTH limit. This can result in SQL or ODBC queries failing to find some records or retrieving truncated data values when the actual data length exceeds the defined column width.

Examples

Right:

The following example illustrates adding a new field "crm_myfield" to "crm_mstr".

It is expected that maximum required size for this field is 32 characters.

```sql
ADD FIELD "crm_myfield" OF "crm_mstr" AS character
DESCRIPTION "Example character field"
FORMAT "x(32)"
INITIAL ""
LABEL "My Field"
MAX-WIDTH 32
MANDATORY
```
See Also

STD-0043 Field Formats Allowed
STD-0064 Character Field Format length limit
STD-0066 Do not use Packed Fields
STD-0301 Oracle MAX-WIDTH (SQL-WIDTH) must be specified on new database fields
STD-0091 Proper Use of Status Line

Summary

Set the status line only to values contained in stline[]. Use ststatus when setting the status line. Do not use "status input off".

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0091</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

In order to present a uniform function key usage to the user, you should use the status lines contained in stline[], a global variable defined in mfdeclre.i. The values of stline[] are set in mf1a.p.

When using status lines, use the global variable ststatus as an intermediate so that when scrolling windows are invoked, the function key usage can be invoked properly. Applhelp.p uses ststatus to capture the present status line setting. It will then reset the status line to that value at the end of the procedure. This allows scrolling help windows to display their own values for the status line.

Do not turn the status input line off. This causes the "status default" line to be displayed instead of the intended line.

All status lines are restricted to 63 characters because that is the maximum the status statement will accept.

As of this writing, stline[] is initialized in mf1a.p with following values:

**Character mode:**

```
stline[1] = "F1=Go 2=Help 3=Ins 4=End 6=Menu 7=Recall 8=Clear 9=Prev 10=Next"
stline[2] = "F1=Go 2=Help 3=Ins 4=End 5=Delete 6=Menu 7=Recall 8=Clear"
stline[3] = "F1=Go 2=Help 3=Ins 4=End 6=Menu 7=Recall 8=Clear 11=Paste"
stline[4] = "F1=Go 2=Help 4=End 7=PgUp 8=PgDn 9=LnUp 10=LnDn"
stline[5] = "F1=Go 2=Help 4=End 7=PgUp 8=PgDn 9=LnUp 10=LnDn Tab=Pivot"
stline[6] = "F1=Go 2=Help 4=End 7=PgUp 8=PgDn 9=LnUp 10=LnDn ENTER=Select"
stline[7] = "F1=Go 2=Help 4=End 7->10=Scroll ENTER=Select Tab=Pivot"
stline[8] = "F1=Go 2=Help 4=End 6=All 7->10=Scroll ENTER=Select"
stline[9] = "F1=Go 2=Help 4=End 6=All 7->10=Scroll ENTER=Select Tab=Pivot"
stline[10] = "F1=Go 2=Help 3=Add 4=End 5=Delete 7=PgUp 8=PgDn 9=LnUp 10=LnDn"
stline[11] = "F1=Go 2=Help 4=End 8=PgDn 9=LnUp 10=LnDn Tab=Pivot"
stline[12] = "F1=Go 2=Help 3=Ins 4=End 5=Detach 6=Menu 7=Recall 8=Clear"
stline[13] = "F1=Go 2=Help 3=Add 4=End 5=Delete"
```

**Windows GUI:**
In general:

- `stline[1]` is used with the `mfnp#.i` include files
- `stline[2]` and `stline[3]` are used within the maintenance programs
- `stline[4]`, `stline[5]`, and `stline[6]` were used in the old scrolling windows (there are some of these still in the product)

The other `stline` extents are used in various specialized 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`

Rationale

Presents a uniform function key usage to the user. Encourages a uniform usage of the status area as a user prompting space.

Examples

Right

Design

Code

```c
ststatus = stline[1].
status input = ststatus.
```

Wrong

Design

Code

```c
status input off.
status default "F1=Go F2=Help F3=Ins F4=End F6=Menu F7=Recall F8=Clear F9=Prev F10=Next".
```
STD-0092 Leap year determination for Y2K compliance

Summary
To determine whether a year is a leap year or not in Progress, use existing Progress "Date" functionality.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0092</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Logic for Dates</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
To determine whether a year is a leap year or not in Progress, use existing Progress "Date" functionality. Do not write your own algorithm to make this determination.

Rationale
The quad-centennial rule for determining leap years is as follows:

Rule 1: Every year that is divisible by 4 is a leap year EXCEPT....
Rule 2: Years divisible by 100, which are not leap years EXCEPT...
Rule 3: Years divisible by 400, which are leap years.

By this definition, 1996 is a leap year (by rule 1) 1900 is not a leap year (by rule 2) 2000 is a leap year (by rule 3)

The Progress DATE function correctly recognizes leap years according to this rule. Therefore, it is preferable to use the date functionality provided by Progress for this purpose. Oftentimes, programmers opt to use a shortcut method based only on the first rule stated above. Although this generally results in the right answer, it is considered bad programming practice to implement only one of the three leap year rules.

Examples
Right

Design
The following illustrates how an include file could be utilized to determine if a year is a leap year. Note that gpleap.i is not part of the MFG/PRO code base, and is shown here only for illustration purposes.

Code
```c
/* gpleap.i – Determines whether a specific year is a leap year * /

Parameters:

/* (1) is an integer value representing a year (i.e. 2000). */
/* This function returns a logical value of true if the input year */
/* is a leap year and false if it is not a leap year */

date(3,1,(1)) - date(2,28,(1)) &gt; 1
```

This is the example shown in the "Wrong" section below, coded correctly. It assumes that the leap year determination is being done by the example include file shown above. The differences between the examples are shown in blue text.

Wrong
/* Calculate whether the fiscal year beginning */
/* in February includes a leap year? */
define variable start_month as integer no-undo.
/* start_month is the starting month for the fiscal year */
define variable start_year as integer no-undo.
/* start_year is the beginning year for the fiscal year */
define variable next_year as integer no-undo.
define variable leapyear like mfc_logical no-undo.
/* Flag used to indicate whether the fiscal year includes a leap year */
assign
leapyear = no
next_year = start_year + 1
leapyear = if (start_month &lt;= 02 and (gpleap.i start_year)) or
           (start_month &gt; 02 and ((gpleap.i next_year))) then
           yes else no.

Wrong

Design

The following illustrates a case in which the leap year is determined based on whether the year is divisible by four. This methodology for determining leap years is incomplete and will result in the wrong answer for input years which are divisible by 100 but not by 400 (for example: 1900 and 2100). The differences between the examples are shown in blue text.

Code

/* Calculate whether the fiscal year beginning */
/* in February includes a leap year? */
define variable start_month as integer no-undo.
/* start_month is the starting month for the fiscal year */
define variable start_year like mfc_logical no-undo.
/* start_year is the beginning year for the fiscal year */
define variable next_year as integer no-undo.
define variable leapyear like mfc_logical no-undo.
/* Flag used to indicate whether the fiscal year includes a leap year */
assign
leapyear = no
leapyear = if (start_month &lt;= 02 and (start_year modulo 4 = 0)) or
           (start_month &gt; 02 and ((start_year + 1) modulo 4 = 0))
           then yes else no.

See Also

N/A
STD-0095 Use BUFFER-COPY to copy records

Summary

Use BUFFER-COPY when copying records within a table or between identical tables, e.g. from a database table to a TEMP-TABLE.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0095</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Table Access &amp; Storage, Field Access &amp; Storage, Data Handling Logic, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities.</td>
</tr>
</tbody>
</table>

Description

When a new record is created by copying another record, all fields in the record must be copied, with possible deliberate exceptions. The statement BUFFER-COPY must be used in these cases. Copy functions are programs such as Item Master Copy, that create a new record and populate it with data they copy from an existing record. You may deliberately skip specific fields, if the copy function you are writing must replace their value in order to achieve its desired result.

Many programs can populate fields in a record, including partner and customer-developed programs. QAD expects that any copy program you write will include the contents of all fields, including double underscore fields (reserved for QAD or for customer use).

BUFFER-COPY will copy all fields in a table. If the table definitions are changed the BUFFER-COPY will make sure all fields are copied and we will avoid the maintenance of copy programs due to DB changes.

Examples

Right

Design

1. Establish the key values for the source and the target records, e.g. PROMPT-FOR.
2. Read the Source record.
3. Perform the BUFFER-COPY, it creates the target record if it doesn't already exist.
4. Assign the key fields for the new record in the BUFFER-COPY ASSIGN statement.

Code

```define buffer new_pt_mstr for file pt_mstr no-undo.
......
prompt-for pt_mstr.pt_part new_pt_mstr.pt_part label "New Item Number".
for first pt_mstr using pt_mstr.pt_part no-lock:
end.
......
buffer-copy pt_mstr except pt_part to new_pt_mstr assign
new_pt_mstr.pt_part.
```

Wrong

Design

A) Explicit assign of field by field.
This is prone to errors when new fields are added to the table.
B) The use of a “null frame”.
This was the old way to do a copy. It still works but it is unnecessarily verbose and impossible to understand for someone unfamiliar with the technique. This method was the old standard and it is frequently used in MFG/PRO. Any occurrences of this method should be replaced as they are encountered.

**Code**

A)  
```plaintext
prompt-for pt_mstr.pt_part new_pt_mstr.pt_part label "New Item Number".
for first pt_mstr using pt_mstr.pt_part no-lock:
end.
create new_pt_mstr.
assign
  new_pt_mstr
  new_pt_mstr.pt_desc = pt_mstr.pt_desc
  new_pt_mstr.pt_desc2 = pt_mstr.pt_desc2
........
  new_pt_mstr.pt__qadc01 = pt_mstr.pt__qadc01
........
  new_pt_mstr.pt__zzzz = pt_mstr.pt__zzzz.
```

B)  
```plaintext
define stream hf.
{mfoutnul.i &stream_name="hf"}
display stream hf pt_mstr with frame hf.
display stream hf part2 @ pt_part with frame hf.
create pt_mstr.
assign pt_mstr.
```

**See Also**

STD-0167 Use TEMP-TABLES instead of xx_wkfl
STD-0097 Avoid using MATCHES in FOR EACH statements

Summary

Performing MATCHES during database retrieval (FOR EACH or SELECT statement) causes a table scan. For this reason, the use of the MATCHES clause is to be avoided. But in instances when MATCHES is required be aware of the impacts on performance and logic.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0097</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Table Access &amp; Storage, Field Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

MATCHES is a powerful tool for string comparison (name search, address search, etc.), but should be used with caution. It has performance impacts and logic quirks that should be known to the designer/developer.

Performance

A retrieval statement like FOR EACH ad_mstr WHERE ad_name MATCHES "QAD" .... causes PROGRESS to perform a table scan of the ad_mstr table. MATCHES does not use index information when performing a comparison; it always scans the entire data set. (reference: PROGRESS Help)

To aid in performance, combine the MATCHES comparison with other WHERE clauses to narrow the result set. Also, place the MATCHES comparison as the last in the list of WHERE clauses. An improved FOR EACH statement is the following:

FOR EACH ad_mstr WHERE ad_state = "CA" and ad_name MATCHES "QAD" ..... This narrows the result set by initially selecting all records for California before looking for matches of QAD.

Right-Trim & Oracle Standard

MATCHES does not ignore trailing blanks as does the equal (EQ) comparison operator. Thus, "QAD" does not match "QAD " although they are considered equal. (reference: PROGRESS Help)

When matching against a database field, enclose the field name in a RIGHT-TRIM( ), TRIM( ) or STRING( ) function. This ensures that Oracle database fields (i.e., fixed length fields) will have the correct comparison. If you fail to use this format, the FOR EACH will return 0 records.

This updates our FOR EACH statement to the following:

FOR EACH ad_mstr WHERE ad_state = "CA" and RIGHT-TRIM(ad_name) MATCHES "QAD" ..... If you fail to use this format, the FOR EACH will return 0 records in Oracle.

Wildcard Characters

MFG/PRO has at least one place where wildcard characters are stored within the data field. This occurs within flh_mstr (Field Help Master), where the flh_call_pgm contains a '*' for pattern matching. In this special case, the order of the comparison values is important. The fixed value needs to be listed 1st and the database field (containing the wildcard) needs to be listed 2nd. If you don't use this format, the FOR EACH will return 0 records.

If the field ad_name contained wildcards, then this updates our FOR EACH statement to the following:

FOR EACH ad_mstr WHERE ad_state = "CA" and "QAD" MATCHES RIGHT-TRIM(ad_name) ..... If you fail to use this format, the FOR EACH will return 0 records.
The WHERE clause in this format will cause additional processing overhead because all rows (records) must be returned to the Progress client session so that the function can be applied against the database field and evaluated.

Rationale

The MATCHES clause is powerful, but can also cause performance and integrity concerns. This example shows the requirement for correctly ordering the database field (containing wildcard) to the right of MATCHES.

Examples

Test91:
FOR EACH flh_mstr WHERE "wlrp01.p" MATCHES TRIM(flh_call_pgm) NO-LOCK:
   DISPLAY flh_call_pgm flh_exec.
END.
   - 14 records found containing wlrp*.p in flh_call_pgm
Expected result

FOR EACH flh_mstr WHERE TRIM(flh_call_pgm) MATCHES "wlrp01.p" NO-LOCK:
   DISPLAY flh_call_pgm flh_exec.
END.
   - 0 records returned  Not the expected result

This example shows the requirement for the TRIM function in the Oracle environment.

Progress Database:

FOR EACH flh_mstr WHERE "wlrp01.p" MATCHES flh_call_pgm NO-LOCK:
   DISPLAY flh_call_pgm flh_exec.
END.
   - 14 records found containing wlrp*.p in flh_call_pgm
Expected result

FOR EACH flh_mstr WHERE "wlrp01.p" MATCHES TRIM(flh_call_pgm) NO-LOCK:
   DISPLAY flh_call_pgm flh_exec.
END.
   - 14 records found containing wlrp*.p in flh_call_pgm
Expected result

Oracle Database:

FOR EACH flh_mstr WHERE "wlrp01.p" MATCHES flh_call_pgm NO-LOCK:
   DISPLAY flh_call_pgm flh_exec.
END.
   - 0 records returned  Not the expected result

FOR EACH flh_mstr WHERE "wlrp01.p" MATCHES TRIM(flh_call_pgm) NO-LOCK:
   DISPLAY flh_call_pgm flh_exec.
END.
   - 14 records found containing wlrp*.p in flh_call_pgm
Expected result

Right

Code
/* THREE SAMPLES OF RIGHT CODE */

/* USE right-trim() FOR ORACLE SUPPORT*/
for each pt_mstr where right-trim(pt_loc) matches "1*" no-lock:
    display pt_part pt_site pt_loc.
end.

/* USE ADDITIONAL CRITERIA TO NARROW TABLE-SCAN SEARCH*/
for each pt_mstr where pt_site = "10000" and
    right-trim(pt_loc) matches "1*" no-lock:
    display pt_part pt_site pt_loc.
end.

/* PLACE WILDCARD CONDITION TO THE RIGHT OF matches */
for first flh_mstr where "wotrorp.p" matches right-trim(flh_call_pgm)
    no-lock:
    display flh_call_pgm flh_exec.
end.

Wrong

Code

/* TWO SAMPLES OF WRONG CODE */

/* MISSING right-trim()*/
for each pt_mstr where pt_loc matches "1*" no-lock:
    display pt_part pt_site pt_loc.
end.

/* WILDCARD CONDITION ON THE LEFT OF matches */
for first flh_mstr where right-trim(flh_call_pgm) matches "wotrorp.p"
    no-lock:
    display flh_call_pgm flh_exec.
end.
STD-0099 Year fields - Validation

Summary

Mandatory Year fields should be validated as within the range defined by the global shared variables low_date and hi_date.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0099</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Data Handling Logic, Logic for Dates</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

The validation of mandatory Year fields should verify that the value entered is within the range defined by the global shared variables low_date and hi_date (i.e. 1900 -> 3999).

Rationale

Since the global shared variables low_date and hi_date specify the range of dates expected to be stored in the database, validating that the year entered in a Year field falls within this range is required. In addition to the use of a properly formatted integer field (see associated standard), this validation will provide additional insurance that a 4 digit year has been specified by the user.

Example

Right

Code

```sql
define variable order_year as integer format "9999" initial 0 no-undo.
...
...
set order_year.
if order_year < year(low_date) or order_year > year(hi_date) then do:
    {mmsg.i 4031 3} /* Invalid year */
next-prompt order_year with frame a.
undo, retry.
end.
```

Wrong

Code

```sql
define variable order_year as integer format "9999" initial 0 no-undo.
...
...
set order_year.
if order_year < 1000 or order_year > 9999 then do:
    {mmsg.i 4031 3} /* Invalid year */
next-prompt order_year with frame a.
undo, retry.
end.
```
See Also

STD-0100 Year fields - Format and initial value
STD-0100 Year fields - Format and initial value

Summary
Year fields should be defined as integer type with a format of "9999" and an initial value of "0".

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields, Character UI Apps, Logic for Dates</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
Year fields should be defined as integer type with a format of "9999" and an initial value of "0". This should not be confused with the year portion of a date field which contains only two digits.

Furthermore, the field help for a Year field should explicitly state that all 4 digits of the year must be entered. This should not be confused with the year portion of a date field which contains only two digits.

Rationale
Proper input to a Year field requires entry of all 4 digits of the year. This is notably different than user input to a variable defined as a date field. For date fields, the user is only required to enter in the last two digits of the year and then the century information is automatically defined and stored based on the user's -yy parameter setting.

Use of an integer format of "9999" for a Year field provides immediate feedback to the end user when less than 4 digits have been entered. For example, if the end user enters 98 into a Year field that has been defined using the "9999" format, the display will show 0098. Formats, such as ">>>9" or ">>>>" do not provide such feedback since the leading zeros will not be displayed and it will be less apparent to the user that they have made an error.

Examples
Right

Code
```
ADD FIELD "cpcd_year" OF "cpcd_det" AS integer
DESCRIPTION "The year associated with this calendar period."
FORMAT "9999"
INITIAL "0"
LABEL "Year"
MAX-WIDTH 4
MANDATORY
```

Wrong

Code
```
ADD FIELD "flp_year" OF "flp_plan" AS integer
DESCRIPTION "The Year on this Family Line"
FORMAT ">>>9"
INITIAL "0"
LABEL "Year"
ORDER 30
```
See Also

STD-0043 Field Formats Allowed
STD-0101 Year fields - Field help
STD-0101 Year fields - Field help

Summary

The field help for a Year field should explicitly state that all 4 digits of the year must be entered.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0101</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Devmstr</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

The field help for a Year field should explicitly state that all 4 digits of the year must be entered. This should not be confused with the year portion of a date field which contains only two digits. Proper input to a Year field requires entry of all 4 digits of the year. This is notably different than user input to a variable defined as a date field. For date fields, the user is only required to enter in the last two digits of the year and then the century information is automatically defined and stored based on the user’s -yy parameter setting.

See Also

STD-0100 Year fields - Format and initial value
STD-0102 Use ASSIGN to Combine Multiple Assignments

Summary
Use a single ASSIGN statement to combine multiple assignments.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0102</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Field Access &amp; Storage, Program Style, Data Handling Logic</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
Combine field and variable assignments into a single ASSIGN statement. Doing many assignments within a single ASSIGN statement is more efficient than issuing one ASSIGN (or doing a simple assignment without the ASSIGN keyword) for each field or variable. In the case where there is a single assignment, it is preferable to use simple assignment without the ASSIGN keyword – but, not mandatory.

Rationale
Increased performance and reduced r-code size.

Examples
Right

Code
```c
assign so_cust = cust
so_bill = cust
so_ord_date = today.
```

Wrong

Code
```c
so_cust = cust.
so_bill = cust.
so_ord_date = today.

/* - or - */

assign so_cust = cust.
assign so_bill = cust.
assign so_ord_date = today.
```
STD-0103 Define Variables NO-UNDO by Default

Summary
By default, variables should be defined with the NO-UNDO option.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0103</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Variable Definition</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
By default, variables should be defined with the NO-UNDO option. This creates a savings in IO processing and thus better performance since the variable's data-writes to the local BI file are unnecessary. If the variable is defined without NO-UNDO then the definition should include a comment to explain how the UNDO processing is used and why this variable must make this exception.

Rationale
Reduced IO, increased performance.

Examples
Right

Code

```qad
Example 1: my_var remains 10 when the user presses the endkey.

define variable my_var as integer no-undo.
my_var = 0.
do transaction on endkey undo, leave:
   my_var = 10.
   update my_var with frame a.
end.
display my_var with frame a.
```

```qad
Example 2: my_var rolls back to 0 when the user presses the endkey.

/* On endkey, my_var will return to original value.*/
define variable my_var as integer. /*Undo*/
my_var = 0.
do transaction on endkey undo, leave:
   my_var = 10.
   update my_var with frame a.
end.
display my_var with frame a.
```

Wrong
/* No no-undo and no comments explaining why not! */

define variable my_var as integer.
my_var = 0.
do transaction on endkey undo, leave:
    my_var = 10.
    update my_var with frame a.
end.
display my_var with frame a.
STD-0104 Use setFrameLabels() function for translating frames into foreign language (eB and up)

Summary

All displayed frames should have setFrameLabels() function called after frame definition.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0104</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Translatable Strings, Procedure Structure, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities (MFG/PRO versions eB and up)</td>
</tr>
</tbody>
</table>

Description

All frames that displays labels or column-labels should have setFrameLabels() function called after the definition of a frame. If the frame doesn't have a definition, the function should be called before the first display of a frame. setFrameLabels() should be called only once for each frame.

setFrameLabels() goes through the widget tree of a frame, determining the fields that belong to the frame and calculating the real estate available for each field's label. setFrameLabels() then sets the correct language version (based on user's language) of the label required for each field.

All fields that display a label within a frame must be supported by Label Detail and Label Master records. Labels and Column labels will not be translated into foreign language if setFrameLabels() is not used.

Examples

Right

Code

Example 1:

```c
form
    addr1 label "Old Address" colon 15
    name1 no-labels
    addr2 label "New Address" colon 15
    name2 no-labels
    with frame a side-labels width 80 attr-space.

/* SET EXTERNAL LABELS */
setFrameLabels(frame a:handle).
```

Example 2:
where ad_sort &gt;= sort and ad_addr &gt;= code ad_type = "c/s_bank"
no-lock by ad_sort with frame b width 80 no-attr-space:

/* SET EXTERNAL LABELS */
setFrameLabels(frame b:handle).
{mfrpchk.1}
display ad_sort format "x(25)"
   ad_addr
   ad_phone
   ad_city format "X(10)"
   ad_state
   ad_zip.
end.

Wrong

Code

Example 1:

form header
   skip(1)
   mc-curr-label et_report_curr skip
   mc-exch-label mc-exch-line1 skip
   mc-exch-line2 at 23 skip(1)
with frame p1 page-top width 132.

/* SET EXTERNAL LABELS */
setFrameLabels(frame p1:handle).
view frame p1.

Example 2:

form
   tt-cmdvrl_div label "Division" at 5
   tt-cmdvrl_rdesc label "Type"
   tt-cmdvrl_gcode label "Group"
   tt-cmdvrl_gdesc label "Description"
with frame f_reldisp title color normal
   (getFrameTitle("APM_RELATIONSHIP_DATA",30))
   no-labels 8 down width 80 attr-space.

/* SET EXTERNAL LABELS */
setFrameLabels(frame f_reldisp:handle).

Example 3:
<table>
<thead>
<tr>
<th>votype[1]</th>
<th>colon</th>
<th>9 no-labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>box[1]</td>
<td>colon</td>
<td>25 no-labels</td>
</tr>
<tr>
<td>min_amt[1]</td>
<td>colon</td>
<td>36 no-labels</td>
</tr>
<tr>
<td>votype[2]</td>
<td>colon</td>
<td>9 no-labels</td>
</tr>
<tr>
<td>box[2]</td>
<td>colon</td>
<td>25 no-labels</td>
</tr>
<tr>
<td>min_amt[2]</td>
<td>colon</td>
<td>36 no-labels</td>
</tr>
<tr>
<td>votype[3]</td>
<td>colon</td>
<td>9 no-labels</td>
</tr>
<tr>
<td>box[3]</td>
<td>colon</td>
<td>25 no-labels</td>
</tr>
<tr>
<td>min_amt[3]</td>
<td>colon</td>
<td>36 no-labels</td>
</tr>
<tr>
<td>votype[4]</td>
<td>colon</td>
<td>9 no-labels</td>
</tr>
<tr>
<td>box[4]</td>
<td>colon</td>
<td>25 no-labels</td>
</tr>
<tr>
<td>min_amt[4]</td>
<td>colon</td>
<td>36 no-labels</td>
</tr>
<tr>
<td>votype[5]</td>
<td>colon</td>
<td>9 no-labels</td>
</tr>
<tr>
<td>box[5]</td>
<td>colon</td>
<td>25 no-labels</td>
</tr>
<tr>
<td>min_amt[5]</td>
<td>colon</td>
<td>36 no-labels</td>
</tr>
</tbody>
</table>

with frame b side-labels width 53 overlay centered row 10.

/* SET EXTERNAL LABELS */
setFrameLabels(frame b:handle).
STD-0106 Inquiries - Mandated program structure

Summary

Every inquiry program must have the structure shown herein.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0106</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style, Reports/Inquiries</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Every inquiry program must have the structure shown below.

The following are important to note:

- Please use mfrpchk.i instead of mfrpexit.i in all programs. While mfrpexit.i calls mfrpchk.i, we wish to eventually obsolete mfrpexit.i.

- mfrpchk.i is never at the same level as mfreset.i, mfrtrail.i, or mftrl080.i. Putting it at the same level would cause the final include file to be skipped if an F4 is pressed or max pages is reached. The final include file is necessary for page scrolling and for cleaning up after the report.

- See referenced Standard "Report - Page Limits and Printing Interruptions" on using mfrpchk.i if you have several levels within your report program.

Rationale

This ensures a standard look and feel for all inquiries.

Examples

Right

Design

Use the following template to design your inquiry 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:
[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|RDS:STD-0031 Report - Page Limits and Printing Interruption] 
 &frm = "a" }

[End data entry statement]

[Begin postprocessing of data entry values]
if (c-application-mode <> 'web') or (c-web-request begins 'data')
then do:

    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 postprocessing of data entry values]

[Assign hi and low criteria ranges...]

[Begin inquiry logic]

/* See gpsselout.i header for explanation of parameters */
{gpsselout.i
 &printType = "terminal"
 &printWidth = 80
 &pagedFlag = " "
 &stream = " "
 &appendToFile = " "
 &streamedOutputToFile = " "
 &withBatchOption = "yes"
 &displayStatementType = 1
 &withCancelMessage = "yes"
 &pageBottomMargin = 6
 &withEmail = "yes"
 &withWinprint = "yes"
 &defineVariables = "yes"
}

/* Actual inquiry logic may vary from simple for each loop */
for each...
    display...
    {mfrpchk.i}
end. /* for each */

{mfreset.i}

/* List Complete */
define variable prod_line like pt_prod_line.
define variable part_type like pt_part_type.
define variable part like pt_part.
define variable descr1 like pt_descr1.

part = global_part.

form
  part
  pt_descr1 no-label
  prod_line
  part_type
with frame a no-underline width 80 attr-space.

/* Set external labels */
setFrameLabels(frame a:handle).

{wbrp01.i}

repeat:

  if c-application-mode <> 'web' then
    update part prod_line part_type with frame a editing:
      if frame-field = "part" then do:
        /* Find next/previous record */
        {mfnp.i pt_mstr part pt_part part pt_part pt_part}
        if recno <> ? then do:
          part = pt_part.
          display
            part
            pt_descr1
          with frame a.
        end.
      end.
    else if frame-field = "prod_line" then do:
      /* Find next/previous record */
      {mfnp.i pt_mstr prod_line pt_prod_line prod_line pt_prod_line pt_prod_part}
      if recno <> ? then do:
        display
          prod_line pt_prod_line pt_prod_part}
pt_prod_line @ prod_line
with frame a.
end.
recno = ?. end.
else do:
status input.
readkey.
apply lastkey.
end.
end.

(wbp06.i &command = update &fields = " part prod_line part_type"
&frm = "a")

if (c-application-mode <> 'web') or
(c-web-request begins 'data') then do:
find pt_mstr where pt_part = part no-lock no-error.
if available pt_mstr then
display
  pt_desc1
  with frame a.
else
display
  "" @ pt_desc1
  with frame a.
hide frame b.
end.

/* Select printer */
(gpseIout.i
 &printType = "terminal"
 &printWidth = 80
 &pagedFlag = "page"
 &stream = ""
 &appendToFile = ""
 &streamedOutputToTerminal = ""
 &withBatchOption = "no"
 &displayStatementType = 1
 &withCancelMessage = "yes"
 &pageBottomMargin = 6
 &withEmail = "no"
 &withWinprint = "yes"
 &defineVariables = "yes")
for each pt_mstr where pt_part >= part
  and (pt_prod_line = prod_line or prod_line = "")
  and (pt_part_type = part_type or part_type = "")
use-index pt_part
no-lock with frame b width 80 no-attr-space:

/* Set external labels */
setFrameLabels(frame b:handle).
[mfrpchk.i]

if frame-line = frame-down and frame-down <> 0
  and pt_desc2 > "" then down 1 with frame b.
display
  pt_part
QAD Development Standards

pt_desc1
pt_um
pt_prod_line column-label {&ppptiq_p_1}
pt_part_type
pt_group column-label {&ppptiq_p_2}.
do with frame b:
down 1.
display
   pt_desc2 @ pt_desc1
   pt_rev @ pt_prod_line
   pt_status @ pt_group.
end.
end. /* for each pt_mstr */

{mfreset.i}
/* List Complete */
(pxmsg.i &MSGNUM=8 &ERRORLEVEL=1)
end.
global_part = part.
{wbrp04.i &frame-spec = a}

See Also

STD-0031 Report - Page Limits and Printing Interruption
STD-0109 Reports - Mandated program structure
STD-0217 Standard output selection routines (eB and up)
STD-0107 Function Keys in MFG-PRO

Summary

This document discusses typical usage of function keys in MFG/PRO.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0107</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Function keys vary depending on the UI environment (Character, Windows GUI, and NETUI).

**Character & NETUI:**
- F1 GO
- F2 HELP
- F3 INSERT
- F4 END
- F5 DELETE
- F6 USER MENU
- F7 RECALL
- F8 CLEAR
- F9 PREVIOUS - scrolling windows only
- F10 NEXT - scrolling windows only
- F11 COPY/PASTE
- F12 COPY/PASTE DEFAULT VALUE

**Windows GUI:**
- F1 HELP
- F2 GO
- F3 INSERT
- F4 No Action
- F5 DELETE
- F6 USER MENU
- F7 RECALL
- F8 CLEAR
- F9 PREVIOUS - scrolling windows only
- F10 NEXT - scrolling windows only
- F11 COPY/PASTE
- F12 COPY/PASTE DEFAULT VALUE
- ESC END

The values of stline[ ] contain the Function Key selections displayed at the bottom of the frame. Refer to STD-0091 (Proper Use of Status Line) for more details.

**Function key information:**

- F1 (GO) is controlled by standard Progress - MFG/PRO does not override this functionality. This applies to F2 when running in Windows mode.
- F2 is the standard HELP function for Progress (F1 in Windows). Requesting Help in MFG/PRO launches applhelp.p.
- F3 (INSERT) is set in mf1a.p as follows to override the Progress default: on f3 insert-mode.
- F4 (END) is controlled by the standard Progress default. (This is done with the Esc key in Windows.)
- F5 (DELETE) is controlled by the standard Progress default but additional processing logic is put in each application program to handle the Delete.
F6 is set to HELP in mf1a.p. The processing of this function key is controlled by applhelp.p.

F7 though F10 default to normal Progress behavior except where overridden in a specific program.

F11 and F12 are set to HELP in mf1a.p. The processing of these function keys is controlled by applhelp.p.

F11 Hitting F11 on a field that contains a value will copy that value into memory. Hitting F11 on a blank field will paste the value within memory into the current field.

F12 Hitting F12 on a field that contains a value will copy that value, along with the field-name, program-name, and your ID into a permanent record (fldf_mstr).

Hitting F12 on a blank field matching the field-name, program-name, and your ID will paste the value from the permanent record into the field. (This would be useful for someone with a repetitive task where some fields are always entered with the same values)

F13 through F39 are also set to HELP in mf1a.p and their processing is controlled by applhelp.p.

Developers should never override the functioning of F1, F2, F4, F5 or F6.

The other keys may be overridden if needed but they should be restored to their original function upon completion or cancellation of the program in which they are changed.

**Rationale**

Consistent function key usage available to the user.

**See Also**

STD-0091 Proper Use of Status Line
STD-0108 Ctrl-, Alt- and Shift-Key Combinations

Summary

This document discusses typical usage of Ctrl-, Alt- and Shift-Key combinations in MFG/PRO

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0108</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Developers may make use of Ctrl-, Alt- and Shift-Key combinations where necessary with the exception of the following:

**Ctrl-Key Combinations already used:**

Ctrl-A Mapped to HELP in mf1a.p - works like F12 key
Ctrl-B Mapped to HELP in mf1a.p - works like F11 key
Ctrl-C Different functions depending on UI:
  - Character & NETUI: Interrupt (Break)
  - Windows UI: Copy function

Ctrl-D Always used like F5 for a DELETE
Ctrl-E Always used like F4 for an END
Ctrl-F Mapped to HELP in mf1a.p - displays the field name via applhelp.p
Ctrl-P Mapped to HELP in mf1a.p - works like F6 key
Ctrl-V Different functions depending on UI:
  - Character & NETUI: No Action
  - Windows UI: Paste function

Ctrl-X Always used like F1 for a GO
Ctrl-F1 Used to access Help About in the Object programs and in all Windows programs

**Alt- and Shift-Key Combinations already used:**

Alt-F1 Used to access the Drill-Down/Power Browses
Alt-F2 Different functions depending on UI:
  - Character & NETUI: No Action
  - Windows UI: Access the Look-Up Browse

Shift-F1 Different functions depending on UI:
  - Character & NETUI non-Object programs: Always used like F1 for a GO
  - Windows UI & Object programs: Access Procedure Help

Rationale

Some combinations are reserved for consistency with other Progress or Windows standards, or are dedicated to certain uses in MFG/PRO.
STD-0109 Reports - Mandated program structure

Summary

Every report program must have the structure shown herein.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0109</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Procedure Structure, Reports/Inquiries</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Every report program must have the structure shown below.

The following are important to note:

- Please use mfrpchki instead of mfrpexit.i in all programs. While mfrpexit.i calls mfrpchki.i, we wish to eventually obsolete mfrpexit.i.

- mfrpchki cannot exist within the same block level as mfreset.i, mfrtrai.i, or mfrt080.i. Putting it at the same level would cause the final include file to be skipped if an F4 is pressed or maximum pages is reached, and may also result in error messages such as "*** terminal already has a conflicting use" or "Paged terminal streams may not be mixed with non paged". The final include file is necessary for page scrolling and for cleaning up after the report.

- See referenced Standard "Report - Page Limits and Printing Interruptions" on using mfrpchki.i if you have several levels within your report program.

- If this is a Full GUI Report, then only one set/update statement (of frame a) is allowed and you MUST use 'frame a'.

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

- Don't put any frame-phrase (WITH) statements on the mainloop REPEAT.

- Don't bundle the different sections of the report, initialize, data entry and batch quoting, with DO or REPEAT blocks.

- Don't use VALIDATE in the data entry statement, validate the data as a part of the post processing instead. The GUI converter can't handle the VALIDATE statement.

Rationale

This ensures a standard look and feel for all reports. It also makes sure that the structure of reports conforms to what is expected by the GUI converted, so that the report can be converted into a Full GUI report.

Examples

Use the following template to design your report program.

Right

Design

{mfdtitle.i "99 "}
/* 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).

/* 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.

{wbrp01.i}    /* General web report setup */
{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 gpselout.i header for explanation of parameters */
{gpselout.i}
 &printType = "printer"
 &printWidth = 132
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 linel label {t001.i} colon 49 skip
  type colon 15 typel label {t001.i} colon 49 skip
  abc colon 15 abcl label {t001.i} colon 49 skip
  site colon 15 sitel 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 linel = hi_char then linel = "".
  if typel = hi_char then typel = "".
  if abcl = hi_char then abcl = "".
  if sitel = hi_char then sitel = "".

if c-application-mode <> 'web':u then
  update
    part
    part1
    line
    linel
    type
    typel
    abc
    abcl
    site
    sitel
    inc_zero_qty
  with frame a.

(wbrp06.i &command = update &fields = "part part1 line linel type typel abc abcl site sitel 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 linel }
  {mfquoter.i type  }
  {mfquoter.i typel }
  {mfquoter.i abc   }
  {mfquoter.i abcl  }
  {mfquoter.i site  }
QAD Development Standards

```sql
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 */
(gpsselout.i
    &printType = "printer"
    &printWidth = 132
    &pagedFlag = "page"
    &stream = ""
    &appendToFile = ""
    &streamedOutputToTerminal = ""
    &withBatchOption = "yes"
    &displayStatementType = 1
    &withCancelMessage = "yes"
    &pageBottomMargin = 6
    &withEmail = "no"
    &withWinprint = "yes"
    &defineVariables = "yes")

(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 */

{mfrpchk.i}
end. /* for each in_mstr ... */

{mfrtrail.i}

end. /* repeat: */

{wbrp04.i &frame-spec = a}

Wrong

Design

Code

This example shows the use of certain constructs (in bold) that will cause the report not to be converted properly by the GUI converter:

```
REPEAT WITH frame a side-labels width 80:
  DO on-error undo, retry:
  [Initialize selection criteria...]
  [Data entry Statement]
  set site VALIDATE( ) .... with frame a.
  [Batch Quoting for batched reports and post processing of data entry values]
  END. /* DO on-error undo, retry */
  [Report logic]
  End. /* repeat */
```

See Also

STD-0217 Standard output selection routines (eB and up)
STD-0106 Inquiries - Mandated program structure

220
STD-0196 Source file formatting - Header (pre-eB2)
STD-0207 Reports that update the database
STD-0031 Report - Page Limits and Printing Interruption
STD-0114 Avoid Use of Shared variable

Summary

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.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0114</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Variable Definition, Procedure Structure, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Do not create a shared variable for the purpose of sharing information across multiple programs/procedures. If shared variables had already been created for this reason, change them to local variables wherever it is possible and use them as parameters to communicate with other programs or procedures.

The parameter could be of type input, input-output or output as demanded by the procedure design.

Do not send the recid of a record to another procedure to read it again. Instead, send a buffer as a parameter.

Rationale

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.

Examples

Right

Code
soprog1:

define buffer bso_mstr for so_mstr.
define variable soCustomer as character no-undo.

....... important logic

{gprun.i 'soprog2.p'
  "(bso_mstr, /* buffer is always input-output */
   input soCustomer)"
}

.......more logic

soprog2:
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

Wrong

Code

soprog1:

define new shared variable so-recid as recid no-undo.
define new shared variable so-cust as character no-undo.

....... important logic

{gprun.i 'soprog2.p'} /*Notice there are no parameters*/

....... more logic

soprog2:
define shared variable so-recid as recid no-undo.
define shared variable so-cust as character no-undo.

for so_mstr where recid(so_mstr) = so-recid no-lock: end.
for first cm_mstr where cm_addr = so-cust no-lock: end.
if available cm_mstr then do:

......more logic
STD-0115 Datatype conversion between date and character

Summary

This standard describes the proper coding protocols for conversion between date and character fields. These protocols support Y2K compliance.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0115</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Data Handling Logic, Logic for Dates, MFG/PRO Application</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

When a date is stored into a character field (or as part of a concatenated character string), store the date as "YYYYMMDD" and utilize the Progress date functions, DAY, MONTH, and YEAR, for creating the required character substrings.

When converting a character string to a date field, the substrings which contain the month, date, and year information must first be converted to integers and then the DATE function is used to create the date. If only two digits of the year are specified in the character string representation then the century information must be determined based on the -yy setting. A general purpose routine, gpgetyr.p, is available for reconstructing the century information.

Exceptions

Due to limitations in the number of characters allowed for file names (8 characters before the period and filetype specification in older DOS environments), it may be necessary to truncate the year field to only two digits when creating the character strings used for archive file names. To convert a character string date that does not contain century information back into a date field requires knowledge of the user’s -yy setting. Determining the -yy setting can be done using the utility program utdictyr.p or using the Progress session handle, SESSION:YEAR-OFFSET. The general-purpose procedure, gpgetyr.p, is available for ascertaining the century information.

Rationale

In general, it is not recommended to store date information in character fields. However, there are specific instances when this is required within MFG/PRO such as for archive filenames, messages, workfiles which will be sorted by date, and customizations/enhancements (ex: programmer needs to add a new date field to a table but doesn't want to change the schema). In these cases, it is very important that all programmers adhere to the same standard for converting between date fields and character strings.

The recommended format for storing dates in character strings is YYYYMMDD. The year is specified first for sorting purposes. It is important to retain the century information to avoid misinterpretation of the date, particularly in situations when the character field is converted back to a date field. All four digits of the year are also required to get the proper chronological sort order across 20th and 21st century dates. If only the last two digits are specified, 20th century dates would appear to come later in time than 21st century dates.

Using syntax such as 'string(start_date)' to convert a date to a character field will have different end results for different -d date display formats. For example, using a -d setting of mdy will yield "MMDDYY" but a -d setting of ymd will result in "YYMMDD". This makes retrieval and/or sorting of the data very difficult for customers with users who utilize a variety of different date display formats. Furthermore, if no format parameter is included in the string function and the date to be converted falls outside the century window defined by the -yy parameter then the last two digits of the year will be truncated producing an incorrect result. For example, for a -yy setting of 1920, the syntax string(low_date) results in "01/01/19". Whereas, for -yy setting of 1900, the same code results in "01/01/00".

Conversion between date fields and character fields needs to be independent of both the users -yy (century window) and -d (date display) settings. The proper method for converting dates to character strings and vice versa utilizes the Progress date functions (month, day, year, date) and is illustrated in the examples below. Also provided
below are general purpose include files from gpapi.i, which can be used for date to character conversions.

**Examples**

**Right**

**Code**

**Example #1**
The following is an example of code which does the date conversion needed for the archive filename, irrespective of the -d setting.

```plaintext
if archive then do:
define variable filename as character no-undo.
filename = "sa" + trim(substring((string(year(today))),3,2))
        + string(month(today),"99")
        + string(day(today),"99") + ".hist".
display filename.
end.
```

**Example #2**
This is an example of general-purpose procedures for converting a date variable to character string. Excerpted from gpapi.i.

```plaintext
渎/* ******************************************************/
Procedure: gpDate2Char
Purpose: Convert a Progress date into its string representation.
Parameters:

<table>
<thead>
<tr>
<th>name</th>
<th>I/O Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-date</td>
<td>The date</td>
</tr>
<tr>
<td>p-string</td>
<td>The string containing the date</td>
</tr>
</tbody>
</table>

Note:
The string is formatted as YYYYMMDD.

**************************************************************************************/
PROCEDURE gpDate2Char:
define input parameter p-date as date no-undo.
define output parameter p-string as character no-undo.

if p-date = ? then
    p-string = "".
else
    p-string = string(year(p-date),"9999")
               + string(month(p-date), "99")
               + string(day(p-date), "99")
END PROCEDURE. /* gpDate2Char */
```

**Example #3**
This is an example of general-purpose procedures for converting a character string to date fields. Excerpted from gpapi.i.
Procedure: gpChar2Date
Purpose: Convert a string into a Progress date.
Parameters:

<table>
<thead>
<tr>
<th>name</th>
<th>I/O Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-string</td>
<td>I The string containing the date</td>
</tr>
<tr>
<td>p-date</td>
<td>O The date</td>
</tr>
</tbody>
</table>

Note:
The string is formatted as YYYYMMDD.

PROCEDURE gpChar2Date:
define input parameter p-string as character no-undo.
define output parameter p-date as date no-undo.
define variable year-part as integer no-undo.
define variable month-part as integer no-undo.
define variable day-part as integer no-undo.
p-date = ?. /* Assume the data is no good. */
if length(p-string) &lt; 8 then do:
  return.
end.
year-part = integer(trim(substring(p-string,1,4))) no-error.
if error-status: error then return.
month-part = integer(trim(substring(p-string,5,2))) no-error.
if error-status: error then return.
day-part = integer(trim(substring(p-string,7,2))) no-error.
if error-status: error then return.
p-date = date(month-part,day-part,year-part) no-error.
if error-status: error then p-date = ?.
END PROCEDURE. /* gpChar2Date */

Wrong

Code

Example #1

if archive then do:
  assign
    filename = string(today,"999999")
    filename = "sa" + substring(filename,5,2)
      + substring(filename,1,2)
    + substring(filename,3,2) + ".hist".
  display filename.
end.

In this example, the programmer is attempting to create an archive file with the name: saYYMMDD.hst. However, this code only produces the intended file name when the -d setting is mdy. If the -d setting is ymd then the filename will be saDDYYMM.hst. If two MFG/PRO users accessing the same database elect to use different -d settings (Ex: one user in Europe and the second in the U.S.) then the archive filenames created by the two users will cause confusion and cannot be sorted properly.

Example #2

Example for converting a character string to date field:
/*Convert char_date into date format.* /
if trim(substring(char_date,3,1)) = "/"
    and trim(substring(char_date,6,1)) = "/"
    then do:
        assign
            dd_mon = integer(trim(substring(char_date,1,2)))
            dd_day = integer(trim(substring(char_date,4,2)))
            dd_yr  = integer(trim(substring(char_date,7,2))) + 1900
            new_date = date(dd_mon,dd_day,dd_yr).
    end.

In this example, the programmer has erroneously assumed the date is in the 20th century. This conversion will not produce the correct answer for 21st century dates.
STD-0145 Multiple DB - Switching aliases (pre-eB2.1)

Summary

When there is a need to switch from the current database to another database (and back again) in a Multiple database environment, use the standard procedures provided in the gpalias-family.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0145</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress,Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Procedure Structure,Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities in MFG/PRO versions eB2 and below. See the Switching Domains standard for versions eB2.1 and up.</td>
</tr>
</tbody>
</table>

Description

Any transaction data that is site dependent need to be accessed (read, create, update, delete) in the DB where that site resides. The DB is stored on the site record in si_db. If the si_db is blank (si_db = ""), it is assumed that the site is in the current DB and you’re dealing with a single DB implementation. If the site DB is different from your current qaddb (si_db <> global_db) you need to switch the DB.

The gpalias family of code is designed to handle the issues involved with switching DBs, so that you should never have to use the CREATE ALIAS in code other than gpalias. The main issue is that DB switching is different between Progress and Oracle. In Progress you only have to assign the alias "qaddb" to a new DB, while in Oracle you have to switch both the dataholder DB ("qaddb") and the schemaholder DB ("qad"). The gpalias procedures handle these issues. Please, see the comments in the different procedures to determine which one you need to use.

When you use gpalias to switch the DB, it's only the qaddb that is switched. Other connected databases like qadhelp and qadgui are not switched. This implies that we don't support Bolt-On modules, that have a separate DB, in a multi database environment. Bolt-ons are supported for single-DB installations. It also implies that customizations that split certain tables into a secondary DB would neither be supported in a multi DB environment.

When you switch the alias from one DB to another, Progress only applies the new alias names to any new programs invoked after the switch. That means that when you switch qaddb from NewYork to Seattle, the Seattle DB is only accessed from the programs invoked after the switching. The program doing the switch, all its parent programs and any persistent procedures will still interpret the alias qaddb as it was when those programs were invoked, i.e. qaddb assigned to the NewYork DB.

If you use persistent procedures, there is logic written into gprunp.i (include file to call persistent procedures) that will make sure that the persistent procedure is always running against the same DB as your current qaddb.

You need to, explicitly, switch the databases back to what they were before you began the switching. The switching is NOT scope like a record, frame or an output to. If you don’t switch back, the next program the user runs will be against a different DB than the user expects.

Rationale

All the complexity and issues of switching DBs are nicely encapsulated in a handful of procedures. If the switching logic has to change we only have to do it in these few programs.

Examples

Right

Design
1. Make sure you have the originating DB in a field or variable that you don't manipulate with the processing in other DBs.
2. Get the site record for the site you want to process against.
3. Switch the DB to the si_db if necessary.
4. Call the new program that will do the processing in the "other" DB.
5. Switch the DB back to the originating DB, i.e. the DB saved in step 1.

NOTE: global_db is a global variable that always contains the dbname of the current qaddb. gpalias changes the value of global_db whenever you switch DBs.

Code

```plaintext
original_db = global_db.
for each si_mstr no-lock where ......:
    /* Switch to the Inventory site */
    if si_db <> global_db and si_db <> "" then do:
        {gprun.i ""gpalias3.p"" "{si_db, output err-flag}" }
        end.
    {gprun.i ""dosomething.p""}
    /* Set the alias back to the original database */
    if global_db <> original_db then do:
        {gprun.i ""gpalias3.p"" "{original_db, output err-flag}" }
        end.
end.
```

Wrong Design

A) Do not explicitly use the CREATE ALIAS statement. This will create a problem under Oracle.
B) Do not try to read or update tables where the records are supposed to be found in the "other" DB. This will look in the table of the original DB where it typically won't find any records or it will find wrong or obsolete records. You need to call a new program to be able to access tables in the new qaddb.
C) Don't forget to switch back to the originating DB. If you don't switch back, the next program the user runs will be against a different DB than the user expects.

Code

```plaintext
original_db = global_db.
for each si_mstr no-lock where ......:
    /* Switch to the Inventory site */
    if si_db <> global_db and si_db <> "" then do:
        A) CREATE ALIAS QADDB for si_db.
        end.
    B) for each ld_det where ld_site = si_site:
        total_qty = total_qty + ld_qty_oh.
        end.
    C) /* No switching back to original DB */
end.
```

See Also

STD-0323 Switching domains in procedure code
STD-0340 Proper assignment of alias for Progress and Oracle databases
STD-0146 File and Field attributes from meta-schema

Summary

Never access the meta-schema directly to get file or field attributes. Use gpfile.i and gpfield.i instead.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0146</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields, Procedure Structure</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

If you need to get file or field attributes, do not access the meta-schema directly. When using a statement like "find _field where _field-name = ...", Progress only searches the schema of the first connected database (the DB that has the alias DICTDB assigned to it).

If the user has split the schema, the find may fail. Instead use gpfile.i and gpfield.i to find the attributes. These routines will scan all connected databases to find the file name or field name specified and return its attributes.

Access the meta-schema using gpfile.i or gpfield.i. The attributes from the meta-schema are available in variables used by these include files, e.g. file_dump_name or field_data-type.

Examples

Right

Code

A) Note: gpfilev.i - Contains variable definition for gpfile.i.
   gpfile.i - Finds file_name across all connected databases.

   {gpfilev.i}
   {gpfile.i &file_name=so_mstr}
   if file_found then do:
     output to value(file_dump_name).
   end.

B) Note: gpfieldv.i - Contains variables definition for gpfield.i.
   gpfield.i - Finds field_name across all connected databases.

   {gpfieldv.i}
   {gpfield.i &field_name=so_cust}
   if field_found then do:
     if field_data-type = "integer" then do:
       end.
   end.

Wrong

Code
A) find _file where _file-name = "so_mstr".
   output to value(_dump-name).

B) find _field where _field-name = "so_cust".
   if _data-type = "integer" then do:
   end.
STD-0148 Program names and messages resolved at runtime - Possible values must be explicitly stated

Summary

Program names and message numbers that are resolved from variables or expressions at runtime, need to be mentioned explicitly in the code too.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0148</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Procedure Structure, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

The cross reference information for a standard MFG/PRO release could contain unresolved references, because some "run" statements and "message" statements occasionally use variables to be resolved at run-time. These reference can be resolved automatically by using certain coding techniques.

Rationale

We use Progress's xref output to capture all run programs and uses of "mfmsg...i" files so we can trace program and message usage. However Progress cannot resolve what the value of variables may be during runtime so any gprun...i or mfmsg...i that uses a variable for program or message cannot be resolved. These references can be resolved automatically by using certain coding techniques.

Examples

Right

Design

The technique to be used is the insertion of "false" statements into the code, where each "false" statement represents a possible value a variable can have during run-time.

Code

Run Example

For example if you have a program that can call programs by a variable "xyz" and it can have only the following values:

```
abc01.p
abc02.p
abc03.p.
```

Then, the following code sample would be acceptable by this standard:

```
/* This block is added to explicitly reference all sub-programs used. */
if false then do:
  {gprun0.i ""abc01.p""
  {gprun0.i ""abc02.p"
  {gprun0.i ""abc03.p"
end.
{gprun.i "xyz"} /* Original code */
```
**Message Example**

This same technique may be used to resolve a similar situation with messages. For example if you have a program that calls messages using a variable "xyz" that may have the following values:

1000  
1001  
1002

Then, the following code sample would be acceptable by this standard:

```c
/* This block is added to explicitly reference all messages used. */
if false then do:
    {mfmsg0.i "1000"} /* Added to meet the standard */
    {mfmsg0.i "1001"} /* Added to meet the standard */
    {mfmsg0.i "1002"} /* Added to meet the standard */
end.

{mfmsg.i "xyz"} /* Original code */
```

mfmsg0.i contains no executable code. The fact that the include filename starts with "mfmsg" allows xref tool to pick up its usage and add the message to the xref table. You can use the file in place of the other mfmsg..i files to minimize the impact on the size of the .r code. However do not use it to actually try and display a message as nothing will happen.

**Wrong**

**Design**

Omitting the false reference is wrong.
STD-0149 Never abbreviate field or file names

Summary

Field or file names must never be abbreviated in the source code.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0149</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Procedure Structure, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Do NOT use field name abbreviations in the source code.

Rationale

Abbreviated field or file names can be ambiguous with fields or files added in the future. Not using abbreviations also makes it easier for someone else to read your code.

Example

Using "ad_st" for "ad_state" would be ambiguous if we later added "ad_strata" to the database.
STD-0150 Local Variable Naming Conventions

Summary

Use meaningful and descriptive names for local variables

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0150</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Variable Definition, Procedure Structure, Program Style, Variable Definition, Procedure Structure, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

1. When selecting names for local variables, use enough description to distinguish the field from similarly named fields in the software. This will prevent problems occurring with field/procedure help, scrolling windows, code text searches (grep) and cross-referencing. This principle applied to field labels also reduces translation costs.

1.1 Consider the variable "line". In MFG/PRO, this variable could mean things like order line, comment line, page line number, etc. Use a self-explanatory name, albeit longer names to distinguish the variable: sales_line, cmmt_line, line_nbr.

1.2 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.

2. Local variables representing a particular item should be named consistently throughout the system. For example, tax_class, a local variable which could be used as selection criteria on a report, a temporary data holder in a process, etc., 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.

3. Some variable names to be careful of using: acct, addr, cc, cd, class, code, date, desc, id, line, list, nbr, number, part, qty, ref, time, tot, trl, type, weight, wt

4. 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.

Examples

Right

Code

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.
```

4. ```
define variable number-lines-on-order as integer no-undo.
```
Wrong

Code

1.

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

4.

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

See Also

N/A
STD-0151 Frames in Maintenance programs - WIDTH

Summary

Use WIDTH 80 when defining frames in maintenance programs.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0151</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

You should always put a WIDTH 80 on every frame except overlay frames. 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.

Rationale

Even if the fields on the screen "pull" the frame to 80 chars in CHUI, that may not be the case in GUI. You will end up with frames that are not nicely lined up if this standard is not followed.
STD-0153 Printing from a maintenance program

Summary

If you have to print from a maintenance program, put all the output commands in a subprogram.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0153</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Reports/Inquiries, Procedure Structure, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

A program must be strictly a print/inquiry program with a single input frame named “frame a” or a maintenance/data-entry program where several frames are allowed. The difference between the two is that the printing program uses mfselprt.i (or similar) to redirect the output, while the maintenance program has all its output directed to the screen all the time. There shall be no redirection of output within 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. The GUI converter can’t handle mixing maintenance and report output in one program.

Examples

Right

Code

```plaintext
xxpgma.p

....
prompt-for .... with frame dataentry.
....
for .... exclusive-lock:
    update .... with frame dataentry.
end.
....
{gprun.i ""xxpgmb.p"")(input parameters)"

xxpgmb.p

....
{mfselprt.i "printer" 132 }
....
for .... no-lock:
    display .... with frame reportdata.
end.
....
{mfreset.i}
....
```

Wrong
Code

```
xxpgm.p

....
prompt-for .... with frame dataentry.
....
for .... exclusive-lock:
   update .... with frame dataentry.
end.
....
{mfse1prt.i "printer" 132 }
....
for .... no-lock:
   display .... with frame reportdata.
end.
....
{mfreset.i}
....
```
STD-0154 Prompts in batchable programs

Summary

A batchable print program must have one and only one prompt (PROMPT-FOR, SET, UPDATE) when it's running in batch mode.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0154</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Reports/Inquiries, Procedure Structure, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Batchable printing programs, i.e. programs that use mfselbpr.i, must only have one input frame which should be named “FRAME A”. When it runs in batch mode it should use only one SET, UPDATE, or PROMPT-FOR statement. In the case when more than one prompt statement is needed to make the UI “work well”, a special provision has to be made for when the report is run in batch. The solution is to have one prompt statement, containing all input fields, that is invoked only if the report is run in batch.

Rationale

MFG/PRO batch request logic can only handle one input statement when running requested programs in batch. The batch processor produces all the input in one long string and it can therefore only be read by one statement. The variable batchrun is a global variable and is set by the batch processor. It is used in the below example to change the UI behavior when the program is run in batch.

Examples

Right

Design

The batchrun variable is used to change the UI behavior depending on if the program is run on-line or in batch.

Code
Wrong Design

Two input statements used, even in batchmode.

Code

....
form
  costset skip
  site skip
  part part1
with frame a ..... .
.... repeat:
  if not batchrun
    then do:
      set costset with frame a.
      set site part part1 with frame a.
      ....
    end. /* if not batchrun */
  else do: /* It is run in batchmode */
    set costset site part part1 with frame a.
  end. /* It is run in batchmode */
  ....
  /* Quoter, mfselbpr.i and for each ... */
  ....
end. /* repeat */

See Also

N/A
STD-0155 Pop-Up Frames in Maintenance programs - add SPACE(2)

Summary

When a frame has no explicit WIDTH (e.g. small overlay frames), always add a SPACE(2) after the last field on every line.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0155</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Reports/Inquiries</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Rationale

This leaves enough space on the right side of the frame for the GUI decoration rectangle.

Examples

Right

Design

The example below adds space(2) at the end of every line in the overlay frame to leave enough space on the right side of the frame.

Code

```plaintext
form
so_ent_ex colon 15 space(2)
so_fix_rate colon 15 space(2)
with frame setb_sub attr-space overlay side-labels centered row frame-row(b) + 4.
```

Wrong

Design

The example below does not add space(2) at the end of every line in the overlay frame.

Code

```plaintext
form
so_ent_ex colon 15
so_fix_rate colon 15
with frame setb_sub attr-space overlay side-labels centered row frame-row(b) + 4.
```

See Also

STD-0151 Frames in Maintenance programs - WIDTH
STD-0156 Frames in Maintenance programs - Positioning widgets

Summary

When aligning text vertically on a form, use the AT, TO, and COLON keywords to start or end Text in specific columns.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0156</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Reports/Inquiries</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Always use a command that gives a widget a fixed position on the screen to align widgets. Never rely on the widgets positioning relative to each other. DO NOT use hard coded spaces to position widgets on the UI.

Rationale

The characters and spaces are a different widths when a screen uses proportional font, which is typically used in GUI.

Examples

Right

Code

form
   pt_part    colon 30
   pt_desc1   at 52 no-label
   skip
   pt_desc2   at 52 no-label
with frame x width 80.

display
   pt_part
   pt_desc1
   pt_desc2
with frame x.

Wrong

Code
form
pt_part colon 30
pt_desc1 at 52 no-label
skip space(52)
pt_desc2 no-label
with frame x width 80.
display
pt_part
pt_desc1
pt_desc2
with frame x.

See Also

STD-0157 Frames in Maintenance programs - explicit SKIP
STD-0157 Frames in Maintenance programs - explicit SKIP

Summary
Always use explicit SKIPs at the end of frame lines.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0157</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Reports/Inquiries</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
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!

Examples

Right
Design
Say you want to create a frame that looks like:

```
+------------------------------------------------------------------------------+
| This is some text just to fill out space Part Number: ________________________ |
| UM: ______________________________________________________________________ |
+------------------------------------------------------------------------------+
```

Note:
This example may violate other standards, it is here only to illustrate what could happen if this standard isn't followed.

Wrong
Design
In GUI you could end up with something that looks like this, if the SKIP is omitted:
| This is some text just to fill out space Part Number: _________________ |
UM: __ | 
+------------------------------------------------------------------+

Code

```c
&SCOPED-DEFINE pgmname_p_01 "This is some text just to fill out space"
/* MaxLen: Comment: */

......
form
{&pgmname_p_01}
  pt_part
  pt_um
  with frame x width ....
    ....
```

QAD Development Standards
STD-0158 Don’t use VALIDATE.

Summary
Don’t use the VALIDATE construct at all. Validations of input data should be done in procedural code instead.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0158</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Reports/Inquiries</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
Don’t use VALIDATE in the data entry statement. Validations of input data should be done in procedure code. GUI converter can’t handle the VALIDATE statement. Embedding "validate()" functions within frames, causes extra work during code restructuring. Each VALIDATE construct has to be broken out to an internal procedure that can be called from the API.

Examples

Right

Design
Write procedural code to validate input data.

Code
```
define variable date_form as date no-undo.
define variable date_to as date no-undo.
f

date_from colon 20
date_to colon 20
with frame a side-labels width 80.

update date_from date_to with frame a.
if date_to < date_from then
   (mfdmsg.i 273). /* Invalid date. Please re-enter */
.
.
.
.
.
```

Wrong

Design
Don’t use VALIDATE when validating input data.

Code
define variable date_from as date no-undo.
form
  date_from colon 20
  date_to colon 20 validate(input date_to &lt; date_from, "Invalid date. Please re-enter")
with frame a side-labels width 80.

update date_from date_to with frame a.
....
....

See Also

N/A
### STD-0159 Frames in Maintenance programs - Positioning frames vertically

#### Summary

Don't use explicit WITH ROW <line-number> where the line-number is hard-coded.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0159</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Reports/Inquiries</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

#### Description

Make them relative to other frames, do not hard code the row it should appear on. NEVER place frames above line 3! Using ROW to position a frame horizontally circumvents PROGRESS' inherent way of placing frames under each other as long as they fit and that creates more maintenance whenever other frames change. Also, since the GUI converter changes the vertical positioning by removing the two top rows (header and the top "box"-line of the first frame), frames will no longer line up in GUI if ROW has been used explicitly.

Since the GUI converter assumes that the two top lines are removed, placing a frame on those two lines creates a problem for the GUI converter.

#### Examples

**Right**

**Design**

A program with 3 frames: a, b and c. Frame a ends on row 5; frame b shows up on row 6. When it is time to view frame c, hide frame b first then Progress will automatically place frame c on the next available row after frame a, which is row 6.

**Code**

```sql
form
  fielda skip
  fieldb skip
  with frame a width 80.
form
  ..... with frame b...
form
  ..... with frame c...
  .... view frame a.
  view frame b.
  ..... hide frame b.
  view frame c.
```
Wrong

Design

This example forces frame c to appear on the row after frame a, row 6. But in GUI frame a will end on row 3 (two top lines removed) and frame c will start on row 6, i.e. two empty line in between. Additionally, will have to modify the form statement for frame c, if we add or remove lines from frame a, which is unnecessary maintenance.

Code

```
form
  fielda skip
  fieldb skip
  with frame a.

form
  .....#
  with frame b... .

form
  .....#
  with frame c... row 6.
  ....
  view frame a.
  view frame b.
  ....
  view frame c.
```
STD-0160 Frames in Maintenance programs - CHOOSE

Summary

Always use the SCROLL option when defining a CHOOSE frame.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0160</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

The WITH phrase of a CHOOSE frame should always contain the SCROLL option, even if it is not needed because the frame can fit all lines. SCROLL is a key phrase the GUI converter needs to see to size and color the frame correctly.

Examples

Right

Code

```plaintext
form
....
with frame abc SCROLL 1.
....
choose row zyx keys xyz with frame abc.
```

Wrong

Code

```plaintext
form
....
with frame abc. /* No SCROLL mentioned */
....
choose row zyx keys xyz with frame abc.
```

See Also

N/A
STD-0161 Frames in Maintenance programs - FORM statements

Summary

All Fields in an input frame must be defined within a single FORM statement.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0161</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Reports/Inquiries</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

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.

The GUI converter needs to modify the form definitions and can only do so on FORM statements.

Examples

Right

Code

```plaintext
form
    pt_part colon 25
    pt_desc1 at 52 no-label
    skip
    pt_um colon 25
    pt_desc2 at 52 no-label
with frame a width 80.

set
    pt_part
    pt_desc1
    pt_desc2
    pt_um
with frame a.
```

Wrong

Code

```plaintext
set
    pt_part colon 25
    pt_desc1 at 52 no-label
    skip
    pt_um colon 25
    pt_desc2 at 52 no-label
with frame a.
```
See Also

STD-0157 Frames in Maintenance programs - explicit SKIP
STD-0162 What the GUI converter does to Full GUI reports

Summary

These are some of the modifications that the GUI converter does when converting a report to Full GUI.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0162</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Reports/Inquiries</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

The following things are automatically handled:

1. `mfdtitle` is moved up to the top of the file.
2. `FIND's` of global control records are replicated in the report subprocedure.
3. Non-shared buffer definitions are moved to the report subprocedure.

From the start of the program file up to first `REPEAT` statement is considered the Procedure prefix code and is left in place as in-line code.

The first `REPEAT` is removed and the below code is inserted opening the first Internal Procedure. This is why the repeat should have NO options!

```c
/*GUI*/ {mfguirpa.i true "printer" 132 }
/*GUI*/ procedure p-enable-ui:
```

The first and only allowed Update/Set statement ends this Procedure and begins the next with the below code. The field list is trapped for later use.

```c
/*GUI*/ run p-action-fields (input "display").
/*GUI*/ run p-action-fields (input "enable").
/*GUI*/ end procedure. /* p-enable-ui, replacement of Data-Entry GUI*/
```

All code between the preceding SET/UPDATE statement and the beginning of the batch quoting is considered 'general' validation and value adjustments.

Code after the batch quoting is consider the 'field' validation area.

Blocks that begin with "if...condition.. then do:" and contain a NEXT-PROMPT are cut out and make into Field Triggers for the field of the NEXT-PROMPT. They are defined immediately after this p-report-quote internal procedure ends, which is at the point where the "gpselout.i" (or "mfsel?????i") include occurs. The inserted code is:
/*GUI*/ end procedure. /* p-report-quote */
/*GUI - field trigger section */

on leave of ... do: ((sample trigger code))
   assign ....
   if ... do:
      {mfmsg.i ## 3}
      /*GUI next-prompt removed */
      /*GUI undo removed */ return no-apply.
   end.
end.

/ *GUI*/ procedure p-report:
/ *GUI*/ {gpprtpa.i "printer"132}

This Report Block section continues to the end of the file where the final END matching the first REPEAT is
removed and the following code is inserted.

/ *GUI*/ end procedure. / *p-report*/
/ *GUI*/ {mfguirpb.i &flds="<list-of-fields>"}

The mfrtrail.i or mftrl080.i include file is replaced with the 2 include files that accomplish the Print Dialog in the GUI
environment. If a trailer is not printed, and the report uses mfreset.i directly, the GUI include files are inserted
before the mfreset.i.

See Also

STD-0109 Reports - Mandated program structure
STD-0164 Read unique records - 'FIND'

Summary
Always use "FOR FIRST <file-name>: END." when reading unique records no-lock

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0164</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress,Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Table Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
To speed up performance on an Oracle database, it is important to reduce the number database reads the 4GL is actually generating. A good way to do this is to take advantage of the Progress DataServer's look-ahead query option. Doing so will limit the number of select statements generated for a particular 4GL database access statement. On all no-lock queries, be sure to utilize a 4GL statement that will use the "look-ahead" query.

FOR EACH/FOR FIRST/FOR LAST constructs take advantage of the "look-ahead" queries. These constructs should be used instead of FIND/FIND FIRST/FIND LAST. The following example shows how to turn FIND statements into statements that will use "look-ahead" queries.

No benefit is gained by using these constructs together with exclusive-lock.

Rationale
Oracle is set related. The FIND construct produces two select statements to accomplish the operation. The first SQL select statement will select the Progress recids that match a given where clause. The second SQL select statement will do a unique select on the record for the recid value that is returned. Replacing this with a FOR statement, that uses the look-ahead query, will yield the record matching the where clause in a single select statement.

Examples
Right

Code
```
Replace FIND/FIND FIRST/FIND LAST commands
The FIND can be replaced with a FOR statement and take advantage of the look-ahead query:
for first pt_mstr no-lock where pt_part = input pt_part: end.
if available pt_mstr then...
```

Wrong

Code
```
find first pt_mstr no-lock where pt_part = input pt_part no-error.
if available pt_mstr then...
```

See Also
STD-0165 Use OPEN QUERY-GET NEXT instead of FIND FIRST- FIND NEXT
STD-0165 Use OPEN QUERY-GET NEXT instead of FIND FIRST- FIND NEXT

Summary

Always use "OPEN QUERY + GET" when iterating through records with no-lock, instead of using FIND FIRST/FIND NEXT or FIND LAST/FIND PREV.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0165</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress,Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Table Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

To speed up performance on an Oracle database, it is important to reduce the number database reads the 4GL is actually generating. A good way to do this is to take advantage of the Progress DataServer's look-ahead query option. Doing so will limit the number of select statements generated for a particular 4GL database access statement. On all no-lock queries, be sure to utilize a 4GL statement that will use the "look-ahead" query.

DEFINE QUERY/OPEN QUERY code constructs take advantage of the "look-ahead" queries. The following examples explain how to turn FIND statements into statements that will use "look-ahead" queries.

This standard does not provide any benefit when records are read using exclusive-lock.

Rationale

Oracle is set related. The FIND FIRST/FIND NEXT constructs produce two select statements to accomplish the operation. The first SQL select statement will select the Progress recids that match a given where clause. The second SQL select statement will do a unique select on the record for the recid value that is returned. Replacing this with a QUERY statement, using the look-ahead query, will yield the full record set matching the where clause in a single select statement.

Example

Right

Code

```sql
define query q_pt_mstr for pt_mstr no-undo.
open query q_pt_mstr for each pt_mstr no-lock where pt_part &gt;= "123".
get first q_pt_mstr no-lock.
dowhile available pt_mstr:
    display pt_part.
    get next q_pt_mstr no-lock.
end.
```

Wrong

Code

```sql
```
find first pt_mstr no-lock where pt_part &gt;= "123" no-error.
  do while available pt_mstr:
    display pt_part.
    find next pt_mstr no-lock where pt_part &gt;= "123" no-error.
  end.

See Also

STD-0164 Read unique records - 'FIND'
STD-0167 Use TEMP-TABLES instead of xx_wkfl

Summary

Use TEMP-TABLES instead of the old xx_wkfl, e.g. sr_wkfl, gltw_wkfl.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0167</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Table Access &amp; Storage, DB Tables, Procedure Structure, Reports/Inquiries</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Do not use permanently defined DB-tables to temporarily store data within a program. The old MFG/PRO standard, developed before TEMP-TABLEs were invented, dictated that we'd use xx_wkfl records when the amount of data to be stored exceeded the limits of WORK-FILES. TEMP-TABLEs should be used for these occasions instead.

Whenever new code is developed, TEMP-TABLEs should be used.

When existing programs that use xx_wkfl are touched they should be redesigned to use TEMP-TABLEs instead, to eliminate the problems that are inherent with using the xx_wkfl.

Rationale

The workfiles in the DB have been used to temporarily store data within a program or a set of programs and sub-programs. It's not used to persist any data that other users can access, qad_wkfl is an exception.

The use of DB-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 be sure that the xx_wkfl is empty we have to delete any records before processing starts. This creates an unnecessary source of error and extra complexity in each program using a xx_wkfl.

It has occurred that records, through unfortunate events, have been left behind and been subsequently picked up by a program that didn't delete workfile records before processing begins. This creates more or less serious problems because these records were picked up by an unrelated program. One thing these problems have in common is that they are very hard to diagnose and almost impossible to re-create.

Using TEMP-TABLES eliminates all these problems, because the TEMP-TABLE is initiated from scratch each time it's used so you can be sure that it is always empty.

It has also been reported that performance improves when TEMP-TABLES are used instead of persistent DB-tables.

Examples

**Right**

**Design**

Use parameters to pass the TEMP-TABLE to a sub-program, to reduce the coupling between procedures, instead if using SHARED TEMP-TABLEs.
abc.p
{defsrwkfl.i}
......
create t_sr_wkfl.
assign
t_sr_nbr = ......
....
....
{gprun.i ""xyz.p"" = (input-output table t_sr_wkfl)"
....
for each t_sr_wkfl no-lock:
......
end.
defsrwkfl.i
define temp-table t_sr_wkfl
  field t_sr_nbr like so_nbr
  field t_sr_line like sod_line
  field t_sr_site like si_site
....
index t_sr_nbr is unique
  t_sr_nbr
  t_sr_line
  ....

xyz.p
{defsrwkfl.i}
define input-output parameter table for t_sr_wkfl.
....
create t_sr_wkfl.
assign
t_sr_nbr = ....
....

Wrong

Design
Use of xx_wkfl.

Code
......
for each sr_wkfl where sr_userid = mfg_user exclusive-lock:
  delete sr_wkfl.
end.
....
create sr_wkfl.
assign
  sr_userid = mfg_user
  sr_nbr = ....

See Also
N/A
STD-0174 Appropriate use of 'Define ... like' syntax.

Summary

1. 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.
2. Always define COLUMN-LABEL for the variable if your program uses a column label for it.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0174</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Variable Definition</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

1. 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. Two fields that have the same native (Progress) data type do not necessarily represent the same kind of data.

2. Always define "COLUMN-LABEL for the variable if your program uses a column label for it."

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 in to 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.

Examples

Right

Code

1. Defining a temporary variable.

define variable temp_part like pt_part no-undo.

2. Also provide a column label.

define variable replace_part like pt_part label "Replacement Part" column-label "Replacement Part" no-undo.
1. Defining a temporary variable.

define variable some_date like pt_cur_date no-undo.

When used as a short cut to get a date variable where some_date in fact has nothing to do with pt_cur_date is not acceptable. Remember the key is that you are creating a variable that really has the same meaning as the referenced variable!

2. define variable replace_part like pt_part label "Replacement Part" no-undo.

Because pt_part's column-label is defined as "?", this will actually work correctly in development. However when pt_part is translated, they might have to come up with a short column label, in which case it would inherit pt_part's column-label, resulting in an inappropriate label.

See Also

N/A
STD-0175 Put explicit SKIP at the last line when using PUT statement

Summary

The use of the "put" verb should be avoided. If you must use "put" to output to a report, then make sure the last line you output includes the "skip" attribute.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0175</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Reports/Inquiries, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Make sure the last line of the output includes the "skip" option if "put" must be used to output to a report. If "skip" is not included and Progress decides that the next line is the top of the next page, the output for the first line of the page will be spaced over by the amount of the previous put.

The "SKIP (expression) specifies the number of new line you want to output. If you do not use the SKIP option, PUT will not start a new line to the output stream. If you use the SKIP parameter, but do not specify expression (or if expression is 0), PROGRESS starts a new line only if output is not already positioned at the beginning of a new line. Anything output by the "put" verb will not appear in output sent directly to the screen. Therefore, the use of "put" is to be avoided as much as possible. The only time you should find it absolutely necessary to use the put verb is to display something that is wider than any currently defined report column. Even then, you can define an alternate frame with no labels or titles and display the information.

Examples

Right

Code
include 'page' in your template.

form header
  page-number
with frame a page-top.
view frame a.

for each ad_mstr no-lock:
  display ad_name.
  put "Outputting a line with a skip" skip.
end.

Could result in an output like:

1
Name
-------------------
Jeremiah
Outputting a line with a skip
Tommy
Outputting a line with a skip
2
Name
-------------------
......

Wrong

Code

output to printer paged.
form header
  page-number
with frame a page-top.
view frame a.

for each ad_mstr no-lock:
  display ad_name.
  put "Outputting a line without a skip".
end.

Could result in an output like:

1
Name
-------------------
Jeremiah
Outputting a line without a skip
Tommy
Outputting a line without a skip
2 &lt;=== the problem!
Name
-------------------
......

Including "skip" as the last word on the "Put" statement would have resulted in page "2" being properly displayed.

See Also
N/A
STD-0178 Arrays must not be used if you need unique help or scrolling windows for each array element.

Summary

Arrays must not be used if you need unique help or scrolling windows for each array element.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0178</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Reports/Inquiries</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

If you must truly have different field help or scrolling windows for different elements of an array, **DO NOT USE** arrays!

Rationale

Both field help and scrolling window help are dependent upon the name of the field as returned by the Progress "frame-field" function. That name is then used to look up the field in either the field help table or the scrolling window table. Please note that these tables have no provisions for using array elements.

Example

Right

Code

```plaintext
define variable surf-spot as character no-undo.
define variable surfed-it like mfc_logical no-undo.
define variable liked-it like mfc_logical no-undo.

form
  surf-spot
  surfed-it
  liked-it
with frame a width 80.
```

Wrong

Code

```plaintext
define variable surf-spot as character no-undo.
define variable surf-log as logical extent 2 no-undo.

form
  surf-spot
  surf-log[1]
  surf-log[2]
with frame a width 80.
```
With this example, you can NOT have separate field help or scrolling windows for surf-log[1] and surf-log[2].

See Also

STD-0333 Database fields with Extents are not supported
STD-0350 Maximum Record (Row) Size Limit
STD-0179 Do not use the 'Abbreviate Index' option

Summary
Do not use the "Abbreviate Index" option.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0179</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Indexes, DB Fields</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
Do not use the "Abbreviate Index" option. This option causes unwanted results in programming. Specifically, a field that has been abbreviated and used like this:

```
find jcd_mstr using jcd_surf_spot.
```

....is translated into:

```
find jcd_mstr where jcd_surf_spot begins input jcd_surf_spot.
```

If jcd_mstr already had a record with jcd_surf_spot = "Rincon Heights" and you were trying to create a record for jcd_surf_spot = "Rincon", you would always get the "Rincon Heights" record on the find. With our maintenance philosophy of only creating records if you can't already find it, you would never get the chance to create a record for "Rincon".

See Also
N/A
STD-0180 Every table must have a unique index & a primary index (pre-eB3)

Summary

Every table must have a unique index and a primary index. The two indexes can be the same or different indexes.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0180</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>4</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Indexes, DB Fields</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities before version eB3.</td>
</tr>
</tbody>
</table>

Description

There should be a single unique index on every table that consists of the minimum number of key fields to guarantee uniqueness of a record in the table. Do not include fields that are convenient for sorting in this main unique index. Do not allow logically duplicate records to be added to the table based on this main unique index. This does not preclude the addition of other unique indexes.

There should also be a single primary index on every table that contains fields reflecting the most common access of the table's records via business logic. In most cases, the primary index will be the same as the main unique index used above to guarantee record uniqueness. However, the primary index does not have to be unique.

Rationale

This standard ensures sure that no program can accidentally create duplicate records.
Tables also cannot be loaded more than once, because the unique index prevents the loading of duplicate records.
The primary index allows for common indexed access of tables from business logic.

Examples

Right

Code

| Table: jcd_mstr |
| Field: jcd_surf_spot (name of surf spot) |
| Field: jcd_surfed_it (logical for whether spot has been surfed) |
| Index1: jcd_surf_spot (Primary, unique) |
| Field: jcd_surf_spot |
| Index2: jcd_surfed_it |
| Field: jcd_surfed_it |
| Field: jcd_surfed_it |

Wrong

Code
Table: jcd_mstr
Field: jcd_surf_spot (name of surf spot)
Field: jcd_surfed_it (logical for whether spot has been surfed)

Index2: jcd_surfed_it (Unique)
Field: jcd_surfed_it
Field: jcd_surf_spot

If only index2 is defined, then it would be possible to have a surf spot that had been both surfed and not surfed, a logical contradiction.

See Also

STD-0007 UNKNOWN values used in Unique Keys in Indexes Must Be Unique
STD-0061 Make all fields ‘Mandatory’
STD-0327 Every table must have a unique index & a primary index (eB3 and up)
STD-0182 Dates - Display format

Summary

All dates that are defined as a Progress "date" data-type must be defined with the format 99/99/99. All maintenance screens that display and require the user to enter a date should include a 2-digit year display.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0182</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>3</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Logic for Dates, Character UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

All dates that are defined as a Progress "date" data-type should be defined with the format 99/99/99. For consistency throughout the product, all maintenance screens that display and require the user to enter a date should include a 2-digit year display.

The format used to display a 4 digit year date must be dependent on the Progress -d date display parameter. (See the "Four-digit date fields" paragraph in the Rationale section, below.)

Progress v8.2A and above uses a default -yy setting of 1950 (century window of 01/01/1950 to 12/31/2049).

Rationale

It has been a long-standing practice within the MFG/PRO code to format all date fields as 99/99/99. With the emergence of the Year 2000 concerns and ISO standards, there is increasing pressure to adopt a four-digit year input and display format. However, given that all dates stored in the MFG/PRO database include all four digits of the year and the fact that converting all existing date fields to a 4 digit year display format is a non-trivial task, there is no compelling reason to change the existing date format standards.

Dates that need to be changed

There are, however, permissible exceptions to this rule. Fixed assets typically have a lifespan of more than 100 years. Similarly, employee birth dates oftentimes fall outside the century window. In such cases, it is best to utilize a date input/display format of 99/99/9999. The major drawback associated with using a 4 digit year date format is that it creates additional overhead in the code in order to make it independent or robust in relation to the user’s -d date display setting (see Examples section below).

Four-digit date fields

If a date field is formatted as 99/99/9999 and the user selects a -d setting of mdy or dmy there is no display problem. However, if the user specifies any other date display setting (ymd, ydm, dym, or mdy) then the date displayed on the screen will have a non-year field displayed in the 4-digit space. For example, the date February 1, 2003 will be displayed as 02/01/2003 using a -d setting of mdy but will be displayed as 03/02/0001 using a -d setting of ymd. This can be confusing to the user because of the additional leading zeros in the day field. For this reason, all Progress "date" data type field definitions must include a format of only 99/99/99. In order to force the date to display properly in a different date display format (such as 2003/02/01 for the ymd format), additional logic is required in the application code:

- The Progress session handle, “date-format” (session:date-format), returns a 3-character string indicating the display order for date components (year, day, month). The default format is "mdy." This can be changed on a per-session basis with the Progress -d startup parameter.
The -d parameter only controls the order in which the month, day, and year of a date are displayed, not how it is stored in the database. All dates are stored in the standard US date format of mdy, regardless of the -d setting.

- As long as the display form is declared with enough characters to accommodate a 4-digit year, the date-format session handle can be utilized to alter the date display format used in the form (i.e. 9999/99/99, 99/99/9999, or 99/9999/99). The example below illustrates how to accommodate the various date display formats.

**Example**

These examples show how to code the exceptional case.

**Right**

**Code**

```qcode
define variable expire_date as date initial "02/01/2003" no-undo.

form
  expire_date format "99/99/9999"
  with frame yy width 80 centered.

CASE session:date-format:
  when "dmy" or when "mdy" then
    expire_date:format in frame yy = "99/99/9999".
  when "ymd" or when "ydm" then
    expire_date:format in frame yy = "9999/99/99".
  when "myd" or when "dym" then
    expire_date:format in frame yy = "99/9999/99".
END CASE.

update exp_date with frame yy.
```

**Wrong**

**Code**

```qcode
define variable expire_date as date initial "02/01/2003" no-undo.

form
  expire_date format "99/99/9999"
  with frame yy width 80 centered.

update expire_date with frame yy.
```

**See Also**

N/A
STD-0183 Sequence Naming

Summary

Construct sequence names from the table prefix and a sequential number. When adding sequences do not use the format "tablename_seq" (i.e. brw_mstr_seq).

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0183</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Sequences</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

To create a sequence name, use the table prefix followed by an underscore, followed by "sq" and a two digit sequential number that indicates the order the sequence was added to the table.

For example: mrp_sq01

Sequence names can be up to 30 characters long (an Oracle limitation), though in practice they will likely be much shorter since only a table name prefix or abbreviation (in CBF schema) should be used in the sequence name.

⚠️ When adding sequences do not use the format "tablename_seq" (i.e. brw_mstr_seq).

Rationale

The naming convention "tablename_seq" is reserved for use by the Progress Oracle DataServer.

Examples

Right

Code

```sql
ADD SEQUENCE "brw_sq01"
INITIAL 0
INCREMENT 1
CYCLE-ON-LIMIT yes
MIN-VAL 0
MAX-VAL 999999999
```

Wrong

Code

```sql
ADD SEQUENCE "brw_mstr_seq"
INITIAL 0
INCREMENT 1
CYCLE-ON-LIMIT yes
MIN-VAL 0
MAX-VAL 999999999
```
See Also

STD-0059 RECID data type
STD-0011 Maintaining Sequences
STD-0187 Always specify no-lock or exclusive-lock

Summary

Always specify either no-lock or exclusive-lock when retrieving records.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0187</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Table Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Avoid the use of share-locks in PROGRESS by always specifying a lock phrase when retrieving records. Use no-lock when reading records and exclusive-lock when modifying or deleting records. This applies only to database tables. The developer may specify a lock phrase on temp-table or work-tables for information purposes only, but this is not necessary because PROGRESS will always implicitly use an exclusive-lock on these types of tables.

1. No-Lock

The "no-lock" option should be used for any files accessed in report and inquiry programs.

2. Exclusive-Lock

Exclusive-lock should be used for retrieving any record that may be updated or deleted, e.g. maintenance programs to prevent deadly embrace.

3. Share-Locks

3.1 Share-locks are what you get by default when you do not specify the type of lock.
3.2 Only the Progress database supports the concept of shared locks. It is not directly supported in Oracle.

3.3 When the PROGRESS client asks for a record share-lock, what Oracle does is fetch the record with a no-lock. If the record is then changed, the PROGRESS client will try to upgrade the lock from a no-lock to an exclusive-lock. The Data Server will then compare the two records to verify that it wasn’t changed, and rejects the change if it has. Even if nothing has changed, this technique will still cause processing overhead.

3.4 One particular use of shared locks is extremely problematic. You can read a record exclusive-lock and then downgrade it to a share-lock to single thread processes or resources. This cannot be replicated under Oracle.

Rationale

The Oracle database does not support share-locks.

Examples

Right

Code
for each pt_mstr no-lock:
    display
    pt_part
    pt_desc.
end.

for each pt_mstr exclusive-lock:
    delete pt_mstr.
end.

Wrong

Code

for each pt_mstr:
    display
    pt_part
    pt_desc.
end.

for each pt_mstr:
    delete pt_mstr.
end.

This code will cause the pt_mstr to be re-read in Oracle before it is actually deleted to see if the record has been changed.

See Also

N/A
STD-0188 Record Selection Proper Use of Indexes

Summary

Always try to select records where the leading fields of an index are set (via =) to a single value.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0188</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress,Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Indexes,Table Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Follow Progress' index bracketing rules to optimize performance. Code query selections so that the leading fields of an index are set (via =) to a single value; the field-names always need to be on the left-hand side of the conditions.

1. Progress queries are always resolved by the Progress server by using an index that either the programmer specifies with the "use-index" or that Progress picks at compile time. The server can resolve the request up to the point where a field is bracketed. When all the fields in a selection criteria are not in the specified index, bracketing will not occur. Also, if "<>" or "or" or "not" is used, bracketing will not occur.

2. For more information on how Progress selects an index during compilation, please refer to PROGRESS Language Reference manual section on "FOR EACH".

Rationale

Improper specification of a record phrase can have a dramatic impact on performance and also cause compile error in Oracle.

Examples

Right

Code

```sql
for each glt_det no-lock where
   glt_entity = "1000" and
   glt_acct = "1040" and
   (glt_sub &gt;= "100" and glt_sub &lt;= "199"):
end.
```

In this example, the server can resolve the query up to the glt_sub field. So only records meeting the initial criteria are returned by the server.

Wrong

Code

```sql
for each glt_det no-lock where
   (glt_sub &gt;= "100" and glt_sub &lt;= "199"):
end.
```

Because there's no index on glt_det where glt_sub is the first field, the server cannot resolve the query, and therefore returns all records.
See Also

N/A
STD-0189 Date field validation - Required dates

Summary

Describes date-field validation and standard warning/error messages for required dates

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0189</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Logic for Dates, Program Style, Procedure Structure</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Date-field validation for required dates should be performed as shown in the example below.

Error Message:
If the user has not entered a value for the date field, use message 711:

```plaintext
Date Required
```

An error level of 3 is specified so the user is prompted to enter a date into the field.

Coding Style:
Enclose error message text within comment delimiters. Insert after the `mfmsg include` statement for code readability.

Examples

Right

Code

```plaintext
set order_date.

if order_date = ? then do:
   {mfmsg.i 711 3} /* Date required */
   next-prompt order_date with frame a.
   undo, retry.
end.

h3.Wrong
h4.Code
```

Wrong

Code

```plaintext
/* No validation */
```

See Also

N/A
STD-0190 Dates - Adding to or subtracting from

Summary

Algorithms to use when adding to or subtracting from a date.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0190</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress,Java</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Logic for Dates,Procedure Structure</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

The logic used to manipulate and calculate dates must properly account for leap years and the differences in the number of days in each calendar month. This is particularly true when calculating anniversary dates or start/end dates based on other dates stored in the database.

The Progress DATE function returns an error message, "Day in month is invalid", when it encounters an invalid input combination of (month,day,year). Therefore, date calculations must include checks to verify the validity of the combination of the three input parameters.

Use general purpose date related functions:

There are several include files and general purpose date functions which can be used to construct the desired date logic that accommodates both leap years and the variability in the number of days in a month.

- monthend.i: used to determine the date of the last day in the month which contains the given date
- gpleap.i: used to determine if a specific year is a leap year.
- fsdteqty.i: used to calculate the decimal quantity of months between two input dates
- fslpyr.p: used to count leap years within an input date range

Rationale

Programmers must be careful when manipulating dates to insure that an illegal (month,day,year) combination does not result. Just because the value for each individual entry falls within the allowable range for that field (1-12 for month, 1-31 for day, and 32767 B.C. to 32767 A.D. for year) does not mean that the combination is valid. Naturally, this is due to the fact that not every month has the same number of days. For example, to calculate one year from today’s date is not as simple as just adding one to the year field. This will result in a Progress error message when today is a leap day.

Examples

Right

Code

Example 1: Calculation of Start Date of next period given initial Start Date

The following is an example of code to calculate the start of the next period given an initial start date, the duration of the period in months, and the starting day for the next period. Notice how the monthend include file is utilized to adjust the period start date because of the known variations in the number of days in the month.
define variable start-date as date no-undo.
define variable total-month as integer no-undo.
define variable total-year as integer no-undo.
define variable month-end as date no-undo.
define variable result-date as date no-undo.

/* First, determine start date of next period, */
/* then add duration to the input start date. */
assign
total-month = month(start-date) + duration
total-year = year(start-date).

/* Adjust result if it does beyond year end. */
if total-month &gt; 12 then do:
assign
total-year = total-year + truncate((total-month / 12),0)
total-month = total-month modulo 12.
if total-month = 0 then
assign
total-year = total-year - 1
total-month = 12.
end. /* if total-month &gt; 12 */
/* Correct for start date that may be beyond a month's end. */
month-end = {monthend.i date(total-month, 1, total-year)}.
if start-day &gt; day(month-end) then
assign
result-date = date(total-month, start-day, total-year).

Example 2: Proper Calculation of Anniversary Date

The following is an example of how to properly calculate an anniversary date either one year in the future or one year in the past from a given start date:

Wrong Code

The following examples illustrate faulty date logic. In the first two examples, the value for the year field is being incremented or decremented regardless of the month and day. In both cases, the existing logic will fail and result in a Progress error when the month and day combination corresponds to the leap day (02/29).

1) Computing an anniversary date one year from the start date:

```plaintext
anniversary = date(month(start_date), day(start_date), year(start_date) + 1)
```

2) Computing a beginning date one year in the past and an end date two years in the future:
start_date = date(month(current_date), day(current_date),
    year(current_date) - 1).

end_date = date(month(current_date), day(current_date),
    year(current_date) + 2) - 1.

In the next example, the value for the month field is being manipulated without paying attention to the variability in the number of days in each month (i.e. date(09,31,1999) is not a valid date).

3) Calculating an end date of 6 months from a user defined start date. This logic only works correctly when day(begin_date) <= 28.

define variable current-month as integer no-undo.
define variable current-year as integer no-undo.
define variable start_date as date no-undo.
define variable end_date as date no-undo.

assign
    current-month = month(start_date) + 6
    current-year = year(start_date).

if current-month &gt; 12 then do:
    assign
        current-month = current-month - 12
        current-year = current-year + 1
    end_date = date(current-month, day(begin_date), current-year) - 1.
end.

See Also

N/A
STD-0191 Date range validation - Ending date must never precede the starting date

Summary

Date-field validation should verify that the starting date for a date range is less than or equal to the ending date.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0191</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Java</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Logic for Dates</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Date validation is required to ensure that a date range starts before or on the same day as it ends. Validate only if values exist or are known. If a value is not required for the start and/or end date, it is important that any validation for that date field first check to see if the value is known or not. See the example below.

Rationale

This validation is important to avoid execution problems if the user erroneously enters an ending date that is prior to the starting date. It ensures that a time period's beginning or start date occurs prior to or on the same day as the end date (Note: Also used for from/to dates, effective/expiration dates).

Right

Code

The following validation insures that the end date is not prior to the start date. However, values for the two date fields are not required. If no value exists for either date field, then no validation is performed.

```plaintext
define variable start_date as date no-undo.

define variable end_date as date no-undo.

... ...

set
  start_date
  end_date.

if start_date <end;gt; ? and
  end_date <end;gt; ? and
  start_date > end_date then do:
  {mfmsg.i 4 3} /* Start date must be prior to end date */
  next-prompt start_date.
  undo main, retry.
end.
```

Wrong

Code

```plaintext
/* No validation */
```
See Also

N/A
STD-0192 Date range validation - Sequential ranges should not overlap

Summary
When data is required to be unique for a given time period, validation must ensure that date ranges never overlap.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0192</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress,Java</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Logic for Dates,Procedure Structure</td>
</tr>
<tr>
<td>Applicability</td>
<td>Applies to all development activities</td>
</tr>
</tbody>
</table>

Description
In situations where data is required to be unique for a given time period, date ranges must never overlap. Date validation is required to ensure that this does not happen. The ending date of an interval should be smaller than the start date of the next interval. In addition, the start date of an interval should be greater than the ending date of the previous interval.

Error Message:
Use Message 122 ("Date ranges may not overlap") to flag overlapping date range conditions.

Rationale
When it is necessary to store data that is unique to a given time period, a date range will be needed to store that period, and proper validation should ensure the uniqueness of the range.

Examples
Right
Code
define variable member-curr as character no-undo.
define variable union-curr as character no-undo.
define variable start-date as date no-undo.
define variable end-date as date no-undo.

set
  member-curr
  union-curr
  start-date
  end-date.

do for mumstr:
  if can-find(first mumstr where
    mu_member_curr = member-curr
    and mu_union_curr = union-curr
    and mu_start_date &lt;= end-date
    and mu_end_date &gt;= start-date)
    then do:
      {mfmsg.i 122 3} /* Date ranges may not overlap */
      undo, retry.
    end. /* If available mumstr */
  end. /* Do for mumstr */
STD-0193 Validating GL Account Information (pre-eB)

Summary

To Validate GL Account information you must use one of the standard validation routines named in this standard.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0193</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>3</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>MFG/PRO Application</td>
</tr>
</tbody>
</table>

Description

If your program must check the validity of a compound account-code, use the standard validation routine that corresponds to the program type, as indicated below. A compound account-code consists of the combination of an account, sub-account and cost center, with the possibility of a GL allocation code in place of the account.

1. All programs, other than those that fit under item 2 or 3 below: use gpglver1.p.


   gpglver1.p displays an error message and returns an error flag, but does not have undo capability built-in. The calling program must handle undo conditions. You may use the include file gpglver1.i (a front end to gpglver1.p) which provides built in undo capability.

Rationale

The validation of GL account code information involves several checks, and the extent of validation can be affected by system settings, such as the gl_verify flag, and account security. The elements of a compound account-code must be validated both individually, and in combination with each other.

Example

To ensure that your programs perform validations correctly, use the standard validation routine that applies to your program. The program type dictates which routine to use.

Right

Code
define variable valid_acct like mfc_logical no-undo.

{gprun.i "gpglver1.p": "(input ap_acct, input ?, input ap_cc, output valid_acct)" }  
if valid_acct = no then do:  
  next-prompt ap_acct with frame b.  
  undo setb, retry.  
end.  

OR:  
{gpglver1.i &acc=ap_acct &sub=? &cc=ap_cc &frame=b &loop=setb}

Wrong

Code

for first ac_mstr where ac_code = ap_acct no-lock:  
  if not available ac_mstr then do:  
    next-prompt ap_acct with frame b.  
    undo setb, retry.  
  end.  
end.

for first cc_mstr where cc_ctr = ap_cc no-lock:  
  if not available cc_mstr then do:  
    next-prompt ap_cc with frame b.  
    undo setb, retry.  
  end.  
end.

The above code is incorrect because it refers to the ac_mstr and cc_mstr files directly. Instead, the code should use gpglver1.p. By using the standard GL validation routines in your code, you are guaranteed that your programs will perform GL validations in the same way as other MFG/PRO programs.

See Also

STD-0216 Validating GL Account Information (eB and up)
STD-0195 Source file formatting - Location of mfdtitle.i or mfdeclre.i

Summary

When used, {mfdtitle.i} or {mfdeclre.i} must directly follow the source file header.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0195</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Implementation</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style, Procedure Structure, Variable Definition</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

An {mfdtitle.i} or {mfdeclre.i} statement, when necessary, must directly follow the modification comment statements in the source file header, so that it is the first executable (non-comment) in the source file. Variable definition statements should never precede the mfdtitle.i or mfdeclre.i statement.

See the mfdtitle.i source file for proper formatting of parameters to be passed to it.

Rationale

The {mfdeclre.i} include file contains definitions for a number of common shared variables used in MFG/PRO procedures.

The {mfdtitle.i} include file incorporates {mfdeclre.i} as well. Placement of one of these files before the first executable statement assures that these shared variables are available when referenced, preventing Progress errors.

Example

Right

Code
/* 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*
*/
/* REVISION: 7.4 LAST MODIFIED: 12/28/93 by: bcm *H269*
*/
/* REVISION: 7.4 LAST MODIFIED: 09/22/94 by: jpm *GM78*
*/
/* REVISION: 7.4 LAST MODIFIED: 02/03/95 by: srk *H09T*
*/
/* REVISION: 7.4 LAST MODIFIED: 02/23/95 by: jzw *H0BM*
*/
/* REVISION: 7.4 LAST MODIFIED: 03/06/95 by: wjk *H0BT*
*/
/* REVISION: 7.4 LAST MODIFIED: 04/17/95 by: jpm *H0CJ*
*/
/* REVISION: 8.5 LAST MODIFIED: 11/17/97 BY: *J26C* Aruna Patil */
/* REVISION: 8.6 LAST MODIFIED: 05/20/98 BY: *K1Q4* Alfred Tan */

(mfdeclre.i)

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

Wrong

Code
STD-0196 Source file formatting - Header (pre-eB2)

Summary
The header of all source code files, regardless of language, must conform to the format shown here.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0196</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>All</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style, Process</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities in MFG/PRO versions eB and below</td>
</tr>
</tbody>
</table>

Description
The header of every source code file, regardless of language, must be formatted exactly as shown below. The following notation conventions have been used:

- angle brackets <> indicate a placeholder for a mandatory item, which is to be replaced with the actual value (brackets should be removed)
- square brackets [] indicate a placeholder for an optional item, which is to be replaced with the actual value (brackets should be removed)
- an ellipsis ... within a line indicates that the preceding item may be repeated as many times as needed
- an ellipsis on a line alone indicates that the preceding line may be repeated as many times as needed

All items within the header must be formatted as comments, and are shown here using the Progress comment delimiters /* */. If the programming language in use does not support this type of comment delimiter, appropriate comment delimiters should be substituted.

```c
/* <Filename> - <Brief Description> */
/* Copyright 1986-<Year> QAD Inc., Carpinteria, CA, USA. */
/* All rights reserved worldwide. This is an unpublished work. */
/* <Revision/Version Tag> */
/* [QAD Program Token - optional placement] */
... /* */
/* <Detailed Description> */
/* */
/* <Revision History> */
/* */
/* [QAD Program Token - preferred placement] */
...
/* */
<Blank Line>
```

Each item shown above is described in detail below, in the order that it must appear in the header.

- Filename - The filename of the source code file, including language-specific extension.
- Brief Description - A short description of the purpose of the source code file, no more than a few words.
Use mixed case. Appears on the same line as the filename.

- Year - The current year in YYYY format, to be placed as shown in the official QAD copyright statement. This field must be updated (if not the current year) anytime a source file is changed.

- Revision/Version Tag - A Revision or Version tag as used in the revision history (see below). The value in this tag will be automatically generated by either RCS or Continuus CM, and will always contain the current version number of the source code file. Once created, the developer should never alter this line.

- Detailed Description - A long description of the purpose of the source file, possibly multiple lines in length. Should be mixed case as in standard English. The detailed description has two standard formats, one for Java and one for all other languages. The formats are described below.

- Revision History - Records revisions made to the file, including date changed and author, with a separate entry for each ECO. The revision history has two standard formats, one for Java and one for all other languages. The formats are described below.

- QAD program tokens (when needed) - Specific strings that tell the ECO processing system and other utilities how to handle the file. The preferred placement of these tokens on new files is below all commented lines of the header and just before the first executable line of the program. An alternate location for QAD tokens is to leave them at the top of the program just under the copyright statement. (See standards on tokens in the reference section below for more details on the tokens.)

Detailed Description and Revision History Formats (Java)

The detailed description and revision history for Java source code files should follow the format required by the JavaDoc documentation generation utility from Sun Microsystems. Specifically, the source code file must at a minimum contain the information shown below. See the Sun website for more details on JavaDoc (http://java.sun.com/products/jdk/1.2/docs/tooldocs/solaris/javadoc.html).

```java
/* <Version Tag>
 */
/**
 * <Detailed Description Line>
 ... 
 * @author <Author Name>
 */
```

Each item shown above is described in detail below, in the order that it must appear.

- Version Tag - The current version of the file as defined by the Continuus CM tool. The tag must be placed inside of percent signs, and with a space following the colon, as follows: %version: %. Since Continuus is case sensitive, this tag must always be all lower case. Upon check-out, Continuus will update this tag with the current version number for the file, e.g. %version: 1.1%.

- Detailed Description Line - The line(s) containing the long description of the purpose of the source file.

- Author Name - The full name of the person who created this revision of the file.

Detailed Description and Revision History Formats (non-Java)

The detailed description consists of one or more lines of the form:

```java
/* <Detailed Description Line>
 */
```

Where the placeholder Detailed Description Line represents the line(s) containing the long description of the purpose of the source file.

The revision history consists of one or more entries, completed by a comment signifying the end of the section. Each entry is a single line containing one or more name-value pairs joined by colons, and separated by one or more spaces. So, the revision history is formatted as follows:

```java
/* <Name>:<Value> ...
 */
```

Where the placeholders shown are:

- Name - The name of an approved standard QAD revision history tag.
• Value - A value appropriate to the preceding tag.

Standard revision history tags are shown in the table below. In general, tags are required to appear only when needed, and for several tags, the column position of and the order in which they appear is not mandated. There are exceptions to this however, and certain tags are required to appear on all revision lines, or are required to be in certain ordinal positions (not columns) on the line. The details are indicated in the table below.

Although it is not required, when modifying a program, try to align tags in the same ordinal and column positions on all revision lines of that header. This will maximize the readability of the header.

<table>
<thead>
<tr>
<th>Revision History Tag</th>
<th>Description</th>
<th>Value (Format)</th>
<th>Required (Y/N or conditions)</th>
<th>Ordinal Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revision</td>
<td>The version of the file as recorded by the CM tool (see below for details on the use of this tag)</td>
<td>Number (#.#...)</td>
<td>Yes, for any changes made under RCS source code control</td>
<td>1</td>
</tr>
<tr>
<td>Version</td>
<td>The version of the file as recorded by the CM tool (see below for details on the use of this tag)</td>
<td>Number (#.#...)</td>
<td>Yes, for any changes made under Continuus CM</td>
<td>1</td>
</tr>
<tr>
<td>BY</td>
<td>The full name of the person or utility that made the revision</td>
<td>String</td>
<td>Yes</td>
<td>2</td>
</tr>
<tr>
<td>DATE</td>
<td>The date on which the revision was recorded by the CM tool</td>
<td>Date (mm/dd/yy)</td>
<td>Yes</td>
<td>3</td>
</tr>
<tr>
<td>ECO</td>
<td>The ECO number under which the revision was made</td>
<td>String ( &quot;XXXX&quot; ) Note: Blank spaces are required before and after the asterisks in the format.</td>
<td>For any changes made under an ECO</td>
<td>any</td>
</tr>
<tr>
<td>TASK</td>
<td>The Continuus task number under which the revision was made</td>
<td>String</td>
<td>For any changes made under Continuus CM</td>
<td>any</td>
</tr>
<tr>
<td>REL</td>
<td>The release number of MFG/PRO in which the revision first appeared. (Only provided for backward compatibility with releases 9.0 and earlier)</td>
<td>Release (#.#)</td>
<td>Obsolete</td>
<td>any</td>
</tr>
<tr>
<td>TIME</td>
<td>The time of day that the revision was made</td>
<td>Time (hh:mm)</td>
<td>No/Optional (Some tools insert time automatically)</td>
<td>any</td>
</tr>
</tbody>
</table>

Use of the "Revision" and "Version" Tags
Depending on which CM tool you are using (RCS or Continuus), you should follow the steps shown below when working with revision history lines in source code.

If you are using the RCS source code control tool:
The Revision tag at the top of the source header must be placed inside of dollar signs, as follows: $Revision: $. Upon check-in, RCS will insert the current version number after the colon, e.g. $Revision:1.1$. When creating a new revision line, the developer must place the same tag in the appropriate position on the revision history line, using the same format $Revision: $. Upon check-in, RCS will also insert the correct version number there. Important: When the source file is next checked out, the dollar signs ($) must be removed from the previous revision line, or RCS will replace the existing version number on that earlier line.

If you are using the Continuus CM tool:
The version tag at the top of the source header must be placed inside of percent signs, and with a space following the colon, as follows: %version: %. Since Continuus is case sensitive, this tag must always be all lower case. Upon check-out, Continuus will update this tag with the current version number for the file, e.g. %version: 1.1%. When creating a new revision line, the developer must copy this current version number into the Version tag of the new line, e.g. version:1.1. Note that because the Version tag in the revision history is not automatically generated, it is never enclosed in percent signs.

Examples are shown below.
Rationale

The establishment of this uniform format for file headers makes it easy to determine the purpose and history of revisions for any given source file, even if you have not worked with it before. In addition, conformance to this standard will make it possible for tools to parse and extract the information contained in the header for a variety of purposes. The standard copyright statement provides legal protection for the intellectual property of the company.

Examples

Examples of complete source file headers are shown below for each of three languages: Progress, C and Java. These are followed by an example of the life cycle of the “Revision” tag when used in the revision history.

Right

Code

Progress 4GL

This Progress example assumes the use of RCS source code control and the Revision tag. Note that the REL tag is no longer used for changes made after the 9.0 release. When modifying Progress programs that were created prior eB release it is not necessary to make the previous revision tags conform to this standard.

```c
/* sosomtxx.p - Sales Order Custom Maintenance
 */
/* Copyright 1986-2002 QAD Inc., Carpinteria, CA, USA.
 */
/* All rights reserved worldwide. This is an unpublished work.
 */
/* $Revision: 1.2$
 */
/*
 */
/* This subroutine adds doohickey maintenance capabilities to MFG/PRO's
 */
/* standard sales order maintenance functionality.
 */
/*
 */
/* Revision: 1.0  BY: Sol Padune  DATE: 08/02/96  REL: 8.5  ECO: *J13J*
 */
/* Revision: 1.1  BY: Red Taflan  DATE: 05/20/98  REL: 8.6  ECO: *XXXX*
 */
/* $Revision: 1.2$  BY: Huey Grande  DATE: 01/10/00  ECO: *YYYY*
 */
/*
 */
/*V8:ConvertMode=Maintenance
 */
/*V8:RunMode=Windows
```
Use of the "Revision" Tag
Examples of the revision history for non-Java source files when using RCS source code control are shown here at each stage in the source control process.

Revision history following check-out:

/*/ Revision: 1.0  BY: Sol Padune  DATE: 08/02/96  REL: 8.5  ECO: *J13J* */
/*/ $Revision: 1.1$  BY: Red Taflan  DATE: 05/20/98  REL: 8.6  ECO: *XXXX* */

Revision history before check-in:

/*/ Revision: 1.0  BY: Sol Padune  DATE: 08/02/96  REL: 8.5  ECO: *J13J* */
/*/ Revision: 1.1  BY: Red Taflan  DATE: 05/20/98  REL: 8.6  ECO: *XXXX* */
/*/ $Revision: $  BY: Huey Grande  DATE: 01/10/00  ECO: *YYYY* */
Revision history after check-in:

/* Revision: 1.0   BY: Sol Padune   DATE: 08/02/96   REL: 8.5   ECO: *J13J*
 */

/* Revision: 1.1   BY: Red Taflan   DATE: 05/20/98   REL: 8.6   ECO: *XXXX*
 */

/* $Revision: 1.2$ BY: Huey Grande   DATE: 01/10/00   ECO: *YYYY*
 */

See Also

N/A
STD-0197 Use of comments inside brackets of an include statement

Summary

Use caution if comments are added inside the brackets of an include statement, particularly if positional parameters are used. A comment may incorrectly be considered as one of the positional parameters.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0197</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style, Procedure Structure</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Use caution if comments are added inside the brackets of an include statement, particularly if positional parameters are used. A comment may incorrectly be considered as one of the positional parameters. Instead, consider using named parameters, including the comments within the quotes of a positional parameter, or adding comments immediately before the beginning of the code defining the include file.

A comment may incorrectly be considered as one of the positional parameters. This problem was encountered frequently when patch markers were used on lines within include brackets.

Examples

Right

Code
If comments are needed to explain a positional parameter they should be included before or after the include file definition:

```c
{xxx.i field1
  field2
  field3}
/* fields 2 logical field used to control need to calculate Average Cost */
```

Example 2
Adding all comments outside the include file is acceptable:

```c
/* Determine tax amount now */
{gprun.i "txcalc.p" "{input tr_type,
  input ref,
  input nbr "}"
```

Example 3
Use of input/output parameters will allow comments to be added:

```c
{gprun.i "soatpck.p"
  "{input pt_part,
    input soc_atp_enabled, /* Use ATP Enforcement */
    input pt_site,
    output performEnforcementFlag}"
```

Example 4
Use of named parameters will allow comments to be added:

```c
{xxx.i &fielda=anyValue
  &field2=itemType     /* comment here */
  &field3=amountAvailable}
```

Wrong Code

Example 1
Comments added within positional parameters are incorrect

```c
{xxx.i field1
  field2  /*useAverageCost*/
  field3}
```

Progress will assume that the comment is the 3rd positional parameter, instead of the intended field3 variable.

Example 2
Using named parameters will incorrectly pass comments as part of the parameter value if coded as this:

```c
{xxx.i &fielda=anyValue
  &field2="itemType  /* comment here */
  &field3=amountAvailable}
```

In this example field2 would also have the comment included as part of the value of the field2 parameter.

See Also

N/A
**STD-0198 Source file formatting - Patch markers**

**Summary**

Patch comments or "markers" are not to be used.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0198</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>All</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

**Description**

Patch comments or "markers" are not to be used for annotating code. The only place where a patch marker is allowed (in fact required) in a source file is the Revision History in the header section of the source file. With the institution of a configuration management (CM) system, it is no longer necessary to preserve all old code in a source file. This was originally done so that a programmer could see the history of modifications made to a source file. A CM system gives the programmer the ability to see such a history by comparing versions of a source file and highlighting the differences. This is a preferable solution to the problem, since patch markers make source files very difficult to read, and have led programmers to create programs specifically for stripping out such comments.

Additionally, the development tool pxtclean.p will automatically remove patch markers in the body of Progress source files. Nonetheless, it is preferable not to rely upon this tool for removing patch markers.

**Examples**

**Right**

**Code**

```plaintext
update
    vod_taxable
    vod_taxc
with frame tax.
```

**Wrong**

**Code**

```plaintext
/*G633**    update vod_taxable vod_taxc where (not {txnew.i}) */
/*G633*/    update vod_taxable vod_taxc
             with frame tax.
```

**See Also**

STD-0276 Source file formatting - no commented out code
STD-0199 Always use a WHERE clause in queries against a database

Summary

Always use a WHERE clause in queries against a database

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0199</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress,Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Table Access &amp; Storage,Field Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

In Oracle, if a WHERE clause is not included in a query, a FULL TABLE SCAN is executed. This causes tremendous performance problems when issuing a select on a table with hundreds of thousands of records; typically queries against tables like tr_hist, pt_mstr, or in_mstr demonstrate the problem quite readily.

Every FIND, FOR, QUERY statement must include a WHERE clause that provides information for an index scan. If the select statement is a wide open search, provide a WHERE clause that accomplishes the same results but keeps the select on an index scan. Not only will this ensure an index scan on an Oracle database, it will also provide information on what index to use for both a Progress database and an Oracle database.

Rationale

This prevents potentially significant performance problems.

Examples

Right

Code

```sql
for each pt_mstr no-lock where pt_part &gt; "":
end.
```

Wrong

Code

```sql
for each pt_mstr no-lock:
end.

OR

for each pt_mstr no-lock where true:
end.
```

Although this last example includes a where clause, it doesn't provide Progress any information for an index scan, which is effectively the same as having no where clause.

See Also
STD-0200 Avoid ‘NOT’ operator in WHERE clause
STD-0200 Avoid 'NOT' operator in WHERE clause

Summary

Avoid using the NOT operator in a WHERE clause.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Table Access &amp; Storage, Field Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

The NOT operator in a WHERE clause causes FULL TABLE SCANS. This is because the NOT operator will produce the sort on the client side of the application – The query will return all the data and the client will sort out all those records that satisfy the NOT condition.

To enhance performance, avoid using statements like
1. WHERE NOT logical_variable
2. WHERE NOT logical_statement

Rationale

This improves performance.

Examples

Right

**Code**

```sql
for each glt_det no-lock where glt_unb = false:
end.
```

Wrong

**Code**

```sql
for each glt_det no-lock where not glt_unb:
end.
```

See Also

N/A
STD-0204 Naming Text Include Files

Summary

If an include file is used to include blocks of text in a program, the name of the file should be picked up from ECO Maintenance according to the procedure described here.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0204</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Procedure Structure, Process, Environment</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

If an include file is used to include blocks of text in a program, the name of the file should be picked up from ECO Maintenance similar to the way an ECO number is retrieved. The procedure for this is described here.

Steps:

1. Enter ECO maintenance in the corporate database (36.8.1) and create new text include file:

   1.1 Enter 23 from the qaddb main menu. This will log you into another machine where you get a command prompt.

   1.2 Enter your UNIX password if prompted with “Password”.

   1.3 Enter your terminal type, which is vt220 if you’re using Reflection.

   1.4 Enter the word “ecosys” and press return.

   1.5 At the MFG/PRO-like main menu, press return when prompted for a password (i.e. enter a blank password).

   1.6 36.8 will take you to the ECO system.

   1.7 Enter a “T” in the ECO field and hit return. The program will fill in the name of the next available text file. For example, T08X will appear in the ECO field. Before creating the text file it is advisable to check each of the source code directories to ensure that the file name has not already been used.

   1.8 Enter “exit” from the command prompt after exiting the ECO System.

   This will log you off the UNIX prompt.

2. File Detail:

   It is not necessary to enter any file names; however, you may choose to enter the file that will call the text file. Entering the calling program makes it easy to identify where the text file is used and in which release it started. The “T” ECO will never be submitted.

3. Release Detail:

   Irrelevant

4. Comments:

   Under comment type 1 enter the full text that will be put in the include file.

5. Create the txxx.i file.

   The include file should have the standard program name and description/copyright/patch log header. Later modifications to the file should be denoted by an additional line being added to the patch log. The text itself should not be marked. Update the “T” ECO record to reflect any changes.
To position the text use "at" or "to" avoid using "colon", under Progress V7 "colon" will cause a colon to appear even if there is no label. Also, do not use hard coded spaces nor the Progress space function to make the text line up; with proportional fonts a space is smaller than most other characters.

6. Submitting the txxx.i file.

Submit the txxx.i file just as you would any other include file; be sure to include it in the file detail of the actual ECO. Prevents the possibility of the same named text file in two releases with very different contents. Provides the translators an easier way to see complete 'paragraphs' of text to translate and to provide a simpler means to include literal strings which are repeated extensively throughout MFG/PRO, such as "To" and "From".
STD-0206 Use of conditional compilation

Summary

Do not use conditional compilation pre-processor directives (&if, &then, &else, &endif) in MFG/PRO applications.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0206</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Environment, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Conditional compilation directives are reserved for infrastructure items only. These directives consist of the following Progress syntax:

```
&IF expression &THEN [<progress source code block>]
[ &ELSEIF expression &THEN <progress source code block> ]
[ &ELSE <progress source code block> ]
&ENDIF
```

Infrastructure items are programs such as those named according to: gp.*[.]p[i] and mf.*[.]p[i]

If a given problem cannot be solved without recourse to conditional compilation, proceed as follows: a) generalize the problem into a class of similar problems, b) propose a general solution for the class of problem, in the form of a new infrastructure item, c) limit the use of conditional compilation to the new infrastructure item, d) use the new infrastructure to solve the particular problem.

Conditional compilation creates a blind-spot for the compiler and therefore the developer.

Compilers detect problems early, are exhaustive in their checking, and make far fewer mistakes than manual methods. The importance of finding development problems early is well known.

Widespread use of conditional compilation would lead to the creation of many conditional expressions, throughout the product. The truth sets of those expressions will not be fully intersecting (otherwise they could all be factored and merged). The result is a combinatorial explosion of possibilities, each a unique blind-spot. Code that may compile correctly under one setting, may not in another; much worse: may compile correctly, but not work correctly.

By-passing the compiler, even partially, puts a heavier burden on code review and testing. Both of which are expensive.

Example

You want to call the internal procedure my-internal-procedure, which is contained in the source file mypl.p. Before you can make the call, you need to run mypl.p persistently.

Your analysis tells you that a single instance of mypl.p will be sufficient in all cases. Further, you know that in any given program you may need to make many calls to internal procedures of mypl.p.

Based on that, you want to provide a simplified mechanism to make the persistent call to mypl.p. In particular you want to avoid the need to explicitly define a handle variable for mypl.p.

(This hypothetical example assumes that the include file gprunp.i does not yet exist in the product.)

Right

Design
1. Propose the creation of program gprunp.i, a general purpose routine to facilitate the invocation of internal procedures.

2. Use gprunp.i to make the call (my-internal-procedure in mypl.p).

**Code**

```c
/* myprog.p - AN ORDINARY APPLICATION PROGRAM */
[copyright and revision history omitted]
gprunp.i "mypl" "p" "my-internal-procedure"

[gprunp.i uses conditional compilation directives to define the handle variable for mypl.p, and to prevent double definition. This routine is part of the standard product infrastructure.
If this routine did not exist in the product, you could propose it. If you do, you will need to make a good case for the need to add it to the infrastructure.]

Wrong

**Design**

Write a front-end file - mypl_front_end.i - to serve your purpose.

This approach is incorrect because it uses conditional compilation in an ordinary application program.
The functionality in gprunp.i was simplified for this example.

**Code**

```c
/* mypl_front_end.i - AN ORDINARY INCLUDE FILE TO CALL INTERNAL PROCEDURES IN MYPL.P */
[ copyright and revision history omitted ]

/* ARGUMENTS */
/* 1 - internal procedure to run */
/* 2 - parameters, if any, for internal procedure */

&if defined(runp_pp_tag_mypl_p) = 0 &then
    &global-define runp_pp_tag_mypl_p
    define new global shared variable runp_h_mypl_p as handle no-undo.
&endif

/* If persistent procedure is not instantiated, then run in persistently. */
if not valid-handle(runp_h_mypl_p) then do:
    {gprun.i "mypl.p" "persistent set runp_h_mypl_p" }
end.

/* Run the internal procedure within external persistent procedure */
run {1} in runp_h_mypl_p {2}.
```
STD-0207 Reports that update the database

Summary

Database access considerations for reports that update the database.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0207</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Procedure Structure, Reports/Inquiries, Table Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities. Any project changing non-compliant programs should upgrade them to this standard.</td>
</tr>
</tbody>
</table>

Description

This standard describes how to minimize the impact of record locking on other users in reports that update the database. It also covers how to accommodate the needs of customers without full Progress that wish to customize the output from reports that update the database.

Minimizing Record Locking Impact

This is achieved by keeping the transaction scope small and under no circumstances prompting for user input within an active transaction.

Reports that update the database can be divided into two categories, depending on how frequently records that are potentially locked by the report are accessed.

Category 1 reports only lock records that are accessed by programs within the module it belongs to, which usually means that the user running the report is the only user accessing this record at the time (i.e., the risk for record contention is smaller).

Category 2 reports lock records that are accessed by several modules, i.e. likely candidates for record contention (e.g. in_mstr, glt_det).

In category 1 reports, we allow a larger number of records to be locked because the impact on other users is smaller. This is necessary because these reports (typically SO Print, Requisition Print etc.) update a "printed" flag in each document indicating that it has been printed. To make sure that each form is printed at the same time the "printed" flag is updated, we have to make one spool file for every time the database transaction is committed. It is desirable to generate one spool file for the whole range selected. Therefore, reports of this type can have a large transaction scope, mainly because the number of users impacted by the larger transaction scope is small or at least limited (i.e., we're taking a calculated risk). Note that the records updated by the report remain locked for the duration of the report execution.

The issues described in category 1 reports (synchronizing the document's "printed" flag with generating of the print file) are also applicable to category 2 reports. However, the impact of the record locking is so severe in this case that it is more important to minimize the transaction scope.

This is done at the cost of tampering with the integrity of the print file and the "printed" flag. The risk the user is running in this instance is the print file is purged but the records have been updated and committed to the database (i.e., records are flagged as printed but no report comes out) if his session is disconnected while running this report. The user will have to manually reset the "printed" flag and rerun the report.

In both categories, we give the user the ability to run the report without committing the updates to the database (update_yn = No). In this case, the report is run as one large transaction and the updates are automatically rolled back at the end of the report. If feasible, the code should be run as no-lock when update_yn = No, so no updates need to be rolled back.

The report printed for these types of reports should have the word "SIMULATION" printed at the top of every page if the updates are not committed to the database (update_yn = No). In MFG/PRO versions eB and above, if the 'simulation' pre-processor tag (see below) is used in conjunction with the variable update_yn for controlling database updating, mfphead.i will automatically generate the SIMULATION report titling, when needed.
Customizing Reports Without Full Progress

Customers may desire to redesign the output from reports that update the database. If they have not purchased full Progress, they are not permitted to make changes to any source code that modifies the database; only changes involving record retrieval and display are allowed. Therefore, any code modifying database tables should reside in a sub-procedure that is separate from the input selection criteria and the output display.

Rationale

Reports updating the database need to limit the impact and duration of record locking. Customers without full Progress need to have the ability to redesign the output from reports that update the database.

Examples

Right

Code

Category 1
This example removes the user input inside of the active transaction. The user can still run the report in “Simulation mode” (update_yn = No), but the user has to indicate that before the report runs.

```plaintext
set
    nbr
    nbr1
    update_yn label "Update"
with frame a.
....

mainloop:
do transaction:
    for each so_mstr where so_nbr &gt;= nbr and so_nbr &lt;= nbr1
exclusive-lock:
    /* REPORT CODE */
    {gprun.i ""report.p""
    /* UPDATE LOGIC */
    {gprun.i ""update.p"
end.

if not update_yn then
    undo mainloop, leave.
end.
```

Category 2
Each iteration of a “master record” (e.g. so_mstr, po_mstr) is one transaction, limiting the number of records locked at one time to records related to one master record. The user can still run the report in “Simulation mode” (update_yn = No), but the user has to indicate that before the report runs. Running the report in simulation mode should eliminate all database updates. This effect has to be carefully documented in field help.
Wrong Code

Category 1
In this example, the user has the ability to back out at the end of the report. The technique is scoping all the updates into one transaction and then asking the user if he wants to commit or undo the updates. There are two significant problems with this approach. First, there is one large transaction and secondly, the user is prompted for input inside of the active transaction. Both of these issues can cause significant locking issues for other users while this report is running.

Category 2
set
  nbr
  nbrl
with frame a.
....

  for each so_mstr where so_nbr &gt;= nbr and so_nbr &lt;= nbrl no-lock:

    /* UPDATE CODE */
    for each sod_det where sod_nbr = so_nbr exclusive-lock transaction:
      ....
      if not batchrun then do:
        yn = yes.
        {mfmsg01.i 605 1 yn} /* UPDATE QUANTITY PICKED ? */

        if yn then
          sod_qty_pick = qty_pick.
        end.
      end.

    /* REPORT CODE */
    display
      ....
    end.

end.

See Also

STD-0109 Reports - Mandated program structure
STD-0214 Allowed Comparison Operators

Summary

The allowed comparison operators for Progress are "=", "<", "<=", ">", "=>" and "<>".

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0214</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

The allowed comparison operators for Progress are:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>returns a TRUE value if two expressions are equal</td>
</tr>
<tr>
<td>&lt;</td>
<td>returns a TRUE value if the first of two expressions is less than the second expression</td>
</tr>
<tr>
<td>&lt;=</td>
<td>returns a TRUE value if the first of two expressions is less than or equal to the second expression</td>
</tr>
<tr>
<td>&gt;</td>
<td>returns a TRUE value if the first of two expressions is greater than the second expression</td>
</tr>
<tr>
<td>&gt;=</td>
<td>returns a TRUE value if the first of two expressions is greater than or equal to the second expression</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>returns a TRUE value if two expressions are not equal</td>
</tr>
</tbody>
</table>

Do not use the mnemonics "EQ", "LT", "LE", "GT", "GE" or "NE".

Rationale

"EQ", "LT", "LE", "GT", "GE" or "NE" are English language specific, the symbols are international.

Examples

Right

Code
for each item no-lock where cat_page = 0:
    display item_num item_name cat_page.
end.

for each item no-lock where cat_page < 0:
    display item_num item_name cat_page.
end.

for each item no-lock where cat_page > 0:
    display item_num item_name cat_page.
end.

Wrong

Code

for each item no-lock where cat_page eq 0:
    display item_num item_name cat_page.
end.

for each item no-lock where cat_page lt 0:
    display item_num item_name cat_page.
end.

for each item no-lock where cat_page gt 0:
    display item_num item_name cat_page.
end.

for each item no-lock where cat_page le 0:
    display item_num item_name cat_page.
end.

for each item no-lock where cat_page ge 0:
    display item_num item_name cat_page.
end.

for each item no-lock where cat_page ne 0:
    display item_num item_name cat_page.
end.
See Also

N/A
STD-0216 Validating GL Account Information (eB and up)

Summary

Validate GL Account Code fields using the gpglvpl.p procedure library.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0216</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>3</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>MFG/PRO Application</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities (MFG/PRO versions eB and up)</td>
</tr>
</tbody>
</table>

Description

If your program must validate a GL Account Code field, and you are working on MFG/PRO version eB or above, use the procedure library gpglvpl.p.

Starting with version eB, GL Account code fields are: financial account (defined in ac_mstr), sub account (defined in sb_mstr), cost center (defined in cc_mstr, and project (defined in pj_mstr).

Do not write validation code that refers to the master files directly. Instead, use the internal procedures provided in the procedure library.

Refer to the attached document for use instructions: gpglvpl.rtf.tg.zip

Note on MFG/PRO versions eB2.1 and above: With domain enabled versions of MFG/PRO, note that the validation procedure library gpglvpl.p validates GL Account Code fields in the domain stored in the "global_domain" globally scoped variable. The domain code stored in this variable is the domain in which a user is currently working in. This procedure will not validate GL Account codes stored outside the current working domain of the user.

Rationale

What constitutes a valid GL Account Code is subject to a precise definition. This definition is part of the software, and is not entirely trivial. Consequently, the validation process requires detailed knowledge of a number of details.

The library allows you to carry out validations correctly, without having to learn all of the details.

Examples

You want to validate a GL account code, represented by the values of "acct, sub-acct, cc, proj."

Right

Design

Use procedures "initialize" and "validate_fullcode," from gpglvpl.p to perform the validation.

When the call to "validate_fullcode" terminates, the value of the "result" variable indicates whether the GL account code is valid, according to the default (most common) account-code validation rules in MFG/PRO.

Refer to the document attached above for information about validation rules.

Code
Wrong

The code below violates this standard because it refers to the General Ledger master files directly to perform the validation.

Code

```plaintext
/*! Validate account */
for first ac_mstr where ac_domain = global_domain
and ac_code = acct and ac_active no-lock:
    if not available ac_mstr or acct = co_pl then do:
        {mfmsg.i 3052 3} /* Invalid account */
        glvalid = no.
        leave.
    end.
end.

/*! Validate sub-account */
if sub_acct &lt;&gt; "" then do:
    for first sb_mstr where sb_domain = global_domain
    and sb_sub = sub_acct no-lock:
        if not available sb_mstr then do:
            {mfmsg.i 3131 3} /* Invalid sub-account */
            glvalid = no.
            leave.
        end.
        else if not sb_active then do:
            {mfmsg.i 3137 3} /* Inactive sub-account */
            glvalid = no.
            leave.
        end.
    end.
end.
```

See Also

N/A
STD-0217 Standard output selection routines (eB and up)

Summary

The include file gpselout.i should be used to select an output destination. It should be used with mfreset.i.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0217</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Procedure Structure, Reports/Inquiries</td>
</tr>
<tr>
<td>Applicability</td>
<td>MFG/PRO version eB and later. Used for reports, inquiries, browses and programs that require the user to specify the output destination.</td>
</tr>
</tbody>
</table>

Description

gpselout.i has two main responsibilities:

Responsibility 1: Enable selection of output destination and batch ID
- allow user to select output device and batch ID
- validate output device and batch ID

Responsibility 2: Redirect output to the destination selected
- configure destination if it is configurable (i.e. a printer set up in mfg/pro)
- set a stream to output to that destination – the stream will either have been passed in as an argument or if no stream is passed in it will be the default stream for the procedure that includes the file.

gpselout.i manages the first responsibility and then it includes a tightly coupled include file called gpprsout.i to manage the second responsibility.

gpselout.i needs to be used to allow the user to select the output destination and mfreset.i must be used to reset the output destination after the data has been outputted.

The include file gpselout.i is a merged version of the selection and validation logic in the following include files:

- mfselbp2.i
- mfselbpri
- mfselp02.i
- mfselp03.i
- mfselprt.i
- mfselpr1.i

In Release eB, calls to mfselbp2.i, mfselp02.i, mfselp03.i and mfselpr1.i have been replaced by calls to gpselout.i and these files have been obsoleted. Calls to mfselprt.i and mfselbpri have been retained, but these include files make a passthru 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.

Arguments:
See gpselout.i for a detailed description of the arguments

Rationale

gpselout.i was created to provide one standard way to allow a user to specify an output destination. Previous to MFG/PRO eB, many cloned versions of output selection programs existed.

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

```c
/* 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"
}
```

Right

Design

start of report program

call gpselout.i with required arguments

report logic with display statements

call mfreset.i to reset output destination and process output to windows printing & email transmission if selected

Code

```c
/* 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"
}
```

Wrong
Design

It is incorrect to have calls to mfselbpr or mfselprt in code you are developing.

It is incorrect to have more than one call to mfreset.i in a code stream following a call to gpselout.i. Note that this call may be made directly from the program requiring output selection, or it can be made from an include file used by that program. For example, mfrtrail.i calls mfreset.i. It is incorrect to use both mfrtrail.i and mfreset.i

Code

```{mfselbpr.i "printer" "80"}

............  program code ..............

{mfrtrail.i}
{mfreset.i}
```

See Also

N/A
STD-0220 UI Fields with Lookups should not have any 'on leave of' triggers

Summary

UI fields with Lookups should not have any “on leave of” triggers. “On leave of” triggers override the standard behavior of the window.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0220</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

The GUI lookup button is controlled by a generic ‘on leave of’ trigger.

Excerpt from mfdtit1.i:

```
on leave of CURRENT-WINDOW anywhere do:
  do on error undo, leave:
    run q-leave in global-drop-down-utilities.
  end.
  run q-set-window-recid in global-drop-down-utilities.
  if return-value = "ERROR":U then return no-apply.
end.
```

It is the q-leave procedure that destroys the button after leaving the field. This only occurs in any of the character code that has been run through the GUI Converter. It is acceptable to use an editing block to accomplish the after field processing.

Rationale

When a field in a particular frame in the UI has ‘on leave of’ logic, this logic overrides the generic trigger. The end result is that the code is never run to destroy the lookup button. The button does not ‘go away’ and becomes part of the ‘tabbing sequence’.

Example

In the code examples shown below, the RIGHT code uses validation on the y variable after the set has occurred. In the WRONG example, the code uses a "on leave of" trigger to do the validation. If the y variable had been a lookup field, the button would have stayed on the screen after the user had tabbed off of the field.

Right

Code
display x y z with frame a.
editloop:
do on error undo, retry:
set x y z with frame a.
if not {gpvaly.i y} then do:
   {mfgmsg.i 123 3} /* Invalid y entry */
   next-prompt y with frame a.
   undo editloop, retry.
end.
if not {gpvalz.i z} then do:
   {mfgmsg.i 234 3} /* Invalid z entry */
   next-prompt z with frame a.
   undo editloop, retry.
end.
end.

Wrong

Code

{mfdtitle.i}

display x y z with frame a.

on leave of y in frame a do:
   if not {gpvaly.i y} then do:
      {mfgmsg.i 123 3} /* Invalid y entry */
      return no-apply.
   end.
end.
end.

editloop:
do on error undo, retry:
set x y z with frame a.
if not {gpvalz.i z} then do:
   {mfgmsg.i 234 3} /* Invalid z entry */
   next-prompt z with frame a.
   undo editloop, retry.
end.
end.
STD-0221 Never delete a message nor change the context of a message for a specific release

Summary

Never change the context of a message or delete a message from the devmstr database. It is acceptable to change the message for grammatical and spelling errors.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0221</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Environment, Devmstr, Translatable Strings</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Message data stored in the devmstr system is not specific to a particular release. The same message numbers and message text are used for all releases.

Rationale

If a developer removes or changes the context of a message during development of a new release, it is removed for all previous releases and it will cause a problem in both the public environments and potentially in any Service Packs.

Example

The PCC (Product Change Control) module was originally the ECN (Engineering Change Notice) module. All of the messages that referenced the acronyms ECR(Engineering Change Request)/ECN(Engineering Change Notice) would need to be changed to reference the new acronyms PCR(Product Change Request)/PCO(Product Change Order). The ECR/ECN messages still apply to previous releases, so it is not possible to just replace the old acronyms with the new ones. Instead, new message records would need to be created for each of the affected messages.

Right

Design

Example A: (Changing the context of messages)
Add new messages for the PCR/PCO system in devmstr and change all appropriate programs to reference the new messages.

Example B: (Removing obsolete messages)
As part of the Record Locking project, all references to message #650, "Have all quotes printed correctly", have been commented out in the code. Even though this error message is no longer used in the current baseline version it still exists in the devmstr system since it is used by previous releases. This error message should not be deleted even though it has been obsoleted in the current release.

Code

Example A: (Changing the context of messages)
Wrong

Design

Example A: (Changing the context of messages)
Change the existing ECR/ECN messages to PCR/PCO messages in devmstr.

NOTE: This method would require no coding changes, but any backward compatibility with previous versions will be lost.

Example B: (Removing obsolete messages)
Remove the obsolete message number 650 - "Have all quotes printed correctly", from the devmstr database and from the all of the code.

Code

Example A: (Changing the context of messages)

```pascal
if ecm_rel_date &lt;&gt; ? then do:
    /*distributed mod not allowed*/
    /*PATCH* (mfm03.i 5611 4 "distributed"
end.
```

See Also

STD-0071 Location of messages
STD-0222 Different types of messaging logic (pre-eB)

Summary

There are different types of messaging logic available in MFG/PRO. This tip identifies and describes the differences in each of the methods.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0222</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Procedure Structure</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities for MFG/PRO Release 9.0 and earlier</td>
</tr>
</tbody>
</table>

Description

Messaging:

mfmsg.i (message number, severity) - Standard include file for messages. This include file also gprun's gpmg.p.

```c
{mfmsg.i 11} /* Adding new record */
```

GUI Example:

Adding new record.

Character Example:

Adding new record.

mfmsg01.i (message number, severity, variable to update) - Messaging include file for updating logical variables

```c
{mfmsg01.i 11 1 del-yn} /* Please confirm delete */
```

GUI Example:

Please confirm delete

Character Example:

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.

GUI Example:

Please confirm delete

Character Example:
N/A

mfmsg02.i (message number, severity, variable to append onto the end of the message) - Messaging include file for concatenated variables

{mfmsg02.i 310 4 po_nbr} /* Delete not allowed, purchase order exists */

GUI Example:

![Error Message](image1.png)

Character Example:

mfmsg03.i (message number, severity, optional value 1, optional value 2, optional value 3) - This routine substitutes a '#' in the msg_desc with the parameter passed to the routine. Up to 5 parameters can be passed. Variables msg_var1 and msg_var2 will have to be passed to ensure that 2 parameters are always passed.

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

GUI Example:

![Error Message](image2.png)

Character Example:

ERROR: SCHEDULE SO ALREADY EXISTS FOR SHIP-TO C123, SITE MAIN. Please re-enter.

mfmsg04.i (message, severity) - This is different from mfmsg.i, in the fact that it does not ask for a message number, but the message itself and should not be used. This include file violates standard STD-0071.

(work order receipt not allowed for final assy order***3)

GUI Example:

![Error Message](image3.png)

Character Example:

ERROR: WORK ORDER RECEIPT NOT ALLOWED FOR FINAL ASSY ORDER. Please re-enter.

mfmsg07.i (message number, severity, optional value 1, optional value 2, optional value 3, variable to update) - This include file is a combination of mfmsg01.i and mfmsg03.i. The routine will substitute a "#" in the msg_desc with the parameters passed to the routine. The 6th parameter is the name of the variable that will be updated by the message statement. This variable can be any data-type, and does not have to be logical. This include file does not use any buttons as part of the view-as phrase.
QAD Development Standards

/* Year has ended for book.  Recalculate depreciation? */

**Message Update**

Year has ended for book. Recalculate depreciation?  **yes**

**Warning**

**WARNING: ITEM UNDER WARRANTY TO 12/31/99. CHANGE START DATE TO 01/18/00?**

**GUI Example:**

**Character Example:**

**mfm07g.i** (message number, severity, optional value 1, optional value 2, optional value 3, variable to update) - This include file is the same as **mfm07.i** with the following exception, this include file uses the buttons yes-no option.

/* Item Under Warranty to #. Change Start date to # ? */

**GUI Example:**

**Character Example:**

**mfm08.i** (message number, severity, variable used to capture message text) - Captures a message to a variable name defined in the calling program so that the message can be used with the put/display statement in a report without displaying at the bottom of the terminal screen.

/* Supplier bank required with this check form */

display stream rport {aperserr.i} msg1 with frame error down.

**GUI Example:**

**Character Example:**

**mfm09.i** (message number, variable to hold message text with data substitutions, optional value 1, optional value 2, optional value 3) - This routine does not display the message. Instead it saves the resulting message in the variable passed into it.

/* Price list <> PO price (Item PL PO): # # */

**GUI Example:**

**Character Example:**

**mfm10.i** (message number, severity, name of variable in which to store message description, optional value 1,
optional value 2, optional value 3) - Retrieves a message and stores it in the specified variable, with up to 3 string parameters allowed for replacement in message text. The message will NOT display on the screen, making this routine suitable for reports or any other time the calling program wants direct control over the user interface behavior.

```{mfmsg10.i 7412 1_err_msg ***** *****} /* Value must exist in generalized codes. Please re-enter */```

**GUI Example:**

**ERROR: VALUE MUST EXIST IN GENERALIZED CODES. PLEASE RE-ENTER.**

**Character Example:**

**ERROR: VALUE MUST EXIST IN GENERALIZED CODES. PLEASE RE-ENTER.**

**fsmsg.i** (message number, name of variable in which to store message description, optional value1, optional value 2, optional value 3) - Retrieves a message and stores it in the specified variable. Optional value 1 and 2 are used to replace "#" found in the message text. Optional value 3 is appended to the end of the message. The message will not display on the screen. This is very similar to mfmsg10.i, but it does not use a severity.

```{fsmsg.i 2510 unattached-message unattached-db *****} /* Database # not available */ display unattached-message with frame x no-labels.```

**GUI Example:**

**Database qaddb not available**

**Character Example:**

**Database qaddb not available**

**gpmsg.p** (message number, severity) - this routine works the same as mfmsg.i.

```{gprun.i gpmsg.p (1,1)} /* Adding new record */```

**GUI Example:**

**Adding new record:**

**Character Example:**

**Adding new record.**

**gpmsg03.p** (message number, severity, optional value 1, optional value 2, optional value 3) - this routine takes the variables then generated the message from running mfmsg03.i.

```{gprun.i **gpmsg03.p** (1263,3,addr2,site1,**")**} /* Schedule SO already exists for ship-to #, site # */```

**GUI Example:**

**Error**

**ERROR: SCHEDULE SO ALREADY EXISTS FOR SHIP-TO C123, SITE MAIN. Please re-enter.**

**Character Example:**

**ERROR; SCHEDULE SO ALREADY EXISTS FOR SHIP-TO C123, SITE MAIN, Please re-enter.**

**gpmsg08.p** (message number, severity, variable used to capture message text) - captures a message to a variable name defined in the calling program so that the message can be used with the put/display statement in a report without displaying at the bottom of the terminal screen. This routine is the same as mfmsg08.i.
GUI Example:

**WARNING: SUPPLIER BANK REQUIRED WITH THIS CHECK FORM.**

Character Example:

```plaintext
WARNING: SUPPLIER BANK REQUIRED WITH THIS CHECK FORM.
```

**wbgpms01.p** (message number, severity, response button, message title, variable used to concatenated, variable used to replace '#' in the message, reserved variable) - general procedure to send specs for displaying MFG/PRO messages in WEB environment.

```plaintext
{gprun.i **"wbgpms01.p"**(1,1,?,?,?,?,?,?)}
```

GUI Example:

```plaintext
Adding new record.
```

Character Example:

```plaintext
Adding new record.
```

**Severity Description**

1 - normal (This is useful for information type messages. The case of the message is not changed, and no prefix is added)
2 - WARNING (Changes the case of the message to upper-case and adds the prefix "WARNING:" to the message text)
3 - ERROR (Changes the case of the message to upper-case and adds the prefix "ERROR:" to the message text)
4 - ERROR w/o "Please re-enter" (A good example of when to use this is when deleting records. The user would not want to re-enter any data when deleting. This severity also changes the message text to upper-case and adds the prefix "ERROR:" to the message text)

**See Also**

STD-0277 Always use pxmsg.i when calling messages (eB and up)
STD-0223 Messages - Including the Text of the Error Message

Summary

Include the text of a message on the same line as the message statement, or on the previous line of the code.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0223</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

The messaging routines (mfmsg.i, pxmsg.i, gpmsg.p, mfmsg01.i, ...) are passed a parameter with the message number. If the total length of the line does not exceed 80 characters, add a comment following the call with the text of the message in sentence case. If the total length of the line (with the comment) exceeds the 80 characters, put the comment of the message text before the call. Do not split the text of the comment onto different lines. The comment may abbreviated or paraphrased if space is limited as long as the intent of the message is clear in the text.

Rationale

Including the text of the message improves code readability. It allows the developer to see the text of the message on the same line with the message number. This avoids having to access the devmstr system to determine the text. This also aids in debugging the code in cases where an incorrect message number has been used. If the text of the error message in devmstr does not match the intended text specified in the code then the problem can be easily and quickly resolved.

Examples

Right

a. Message text is before the include statement and is in sentence case:

```c
/*PATCH*/ /*Extended Invoice cost is greater than extended PO cost*/
/*PATCH*/ (mfmsg.i 5 3)
```

b. Message text follows the include statement and is in sentence case:

```c
{mfmsg.i 1 1} /* Adding new record */
if not available pt_mstr then do:
    {mfmsg.i 7179 3} /* Item does not exist */
    undo, retry.
end.
```

Wrong

a. Message text follows the include statement and is all caps:
if (svc_que_prog &gt;= 0 and svc_que_prog &lt;= 3) then do:
  end.
else do:
  (mfmsg.i 324 3)
  /* QUEUE MANAGER MUST BE 0, 1, 2, OR 3 */
  next-prompt svc_que_prog with frame call1.
  undo setloop1, retry setloop1.
  end.

b. No message text provided at all:

if search (temp-filename) = ? then do:
  (mfmsg03.i 5022 2 temp-filename "**** ****")
  end.

c. Message text split between successive lines and is all caps:

{mfmsg.i 440 1} /* CHECKING FOR RECORDS WITH THIS CODE. PLEASE WAIT. */

See Also

N/A
STD-0226 Messages - Bells and Pauses

Summary
The proper use of the pause statement when used with messages. The bell statement should not be used in conjunction with messages.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0226</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
Bell statements are redundant when used with messages. The messaging include files already contain the bell statement. By default, messages in the Windows environment automatically perform the default sound (bell). Developers should only use bell statements when its appropriate and not in conjunction with messages.

Pause statements following the messaging logic are only required if there is no other logic which will pause the execution. Some of the messaging include files will do this automatically when they prompt the user to confirm or update a particular variable. In general, most error conditions should result in the user being prompted to enter a different value into the field where the error occurred. In such cases, a pause statement is unnecessary and redundant. Cases where a pause may be required are: following informational type messages that are displayed at the bottom of the screen with no subsequent action required by the user, or in cases where the error condition necessitates a return to the calling procedure.

Pause statements, when used with messaging, should only be used to pause the action of the program in a character environment. By default, messages shown as alert-boxes in the Windows environment automatically stop the action. Nothing else can be performed until the message is dismissed. If the developer needs to suspend the action of a program, do a check on the "(&WINDOW-SYSTEM)" preprocessor-name for "TTY" to determine whether or not to use the pause statement in conjunction with a message (see example below). In addition, if a pause statement is used within a program that will be run using CIM, then the code should be preceded by the conditional, "if not batchrun".

Rationale
Unnecessary pause statements will require multiple, unnecessary responses from the user. The goal is to avoid requiring the end user to hit the Return/Enter key multiples times in response to a single message.

Examples

Right

Code
Example 1 (bell):

    if po-attached then do on endkey undo, retry:
    confirm_undo = no.
    {mmsg01.i 12511 confirm_undo} /* Yes to undo; No to edit PO list */
    end.
else
    confirm_undo = yes.

Example 2 (pause):
Wrong Code

Example 1 (bell):

```java
if po-attached then do on endkey undo, retry:
    confirm_undo = no.
    bell.
    {mfmsg01.i 12511 confirm_undo} /* Yes to undo; No to edit PO list */
end.
else
    confirm_undo = yes.
```

Example 2 (pause):

```java
if error_flag then do:
    {mfmsg.i 81293} /* Not a valid customer or ship-to */
    pause.
    return.
end.
```

See Also

N/A
STD-0227 Maximum acceptable Source Line Length - 80 characters

Summary

Limit the source code line length to 80 characters.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0227</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>All</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style, Translatable Strings</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Limit the source code line to 80 characters for easy reading and cleanliness of the code. It will help the developer and other developers debugging the code if each line is kept under the 80 character limit.

Exception:
Translatable string preprocessors (in use until eB2) are the only exception to this guideline. They are allowed to be longer than 80 characters because of the way Progress handles the substitution.

Example: These examples do not need to be split onto 2 lines because of the way Progress handles preprocessors.

```
&SCOPED-DEFINE ecmt03_p_3 "WARNING: LISTING TRUNCATED DUE TO WORKFILE LIMIT."
&SCOPED-DEFINE arsbrp03_p_15 ~ "Self-bill Adjustment Lines Pegged To This Invoice Or Memo"
```

The rationale is to promote consistent formatting and readability of code. No matter what editor or UNIX command is used to 'view' the file, there will not be any line wrapping. It also makes debugging easier as no code is 'lost' in the scroll area. Several of the internal tools can be 'broken' by excessively long lines.

Right Code
Wrong

Code

```plaintext
/* ********** Begin Translatable Strings Definitions ********** */
&SCOPED-DEFINE sosoisu3_p_1 "Effective"
/* MaxLen: Comment: */
/* ********** End Translatable Strings Definitions ********** */

/*----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8*/
/* EACH OF THESE STATEMENTS ARE LONGER THAN 80 CHAR. */

define new shared variable eff_date like glt_effdate label {&sosoisu3_p_1} no-undo.
for first sod_det where sod_nbr = scx_nbr and sod_line = scx_line and sod_part = scx_part no-lock: end.
```

See Also

N/A
STD-0228 Proper formatting of comments

Summary

All comment lines should be demarcated and can be in either sentence case or uppercase.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0228</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

When adding comments to the source code, the format should be to have a

```*/ * /
```

when it is a single line comment. When comments span multiple lines, it can either keep the single line comment format

```*/ * /
```

on every line. It is also acceptable to create a block comment with an asterisk at the beginning of each comment line that is not the first or last line of the block, such as:

```/*
  OF A BLOCK COMMENT
  THAT SPANS MULTIPLE LINES. */
```

Comments should be at the same indentation level as the code.

This development standard should not be confused or related to the issue of commenting out code because of a patch. This development standard ONLY applies to the developer adding explanatory comments to the code.

Rationale

Comments need to be easily discernible within the context of the source file and when doing a grep.

Examples

Right

Code

Example 1: Note indentation and multi-line comment formatting.

```*/ THE INPUT PARAMETER TO sosomt1.p, NO, MEANS "NO, THIS IS */
*/ NOT AN RMA" TO THAT PROGRAM. */

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

Example 2: Note indentation and multi-line comment formatting.
```c
/* THE INPUT PARAMETER TO sosomt1.p, NO, MEANS "NO, THIS IS
 * NOT AN RMA" TO THAT PROGRAM. */

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

Wrong

Indentation is not aligned with code and second line has no */ or * to indicate the line is a comment when viewed out of context.

```c
/* THE INPUT PARAMETER TO SOSOMT1.P,
 NO, MEANS, "NO, THIS IS
 NOT AN RMA" TO THAT PROGRAM. */

{gprun.i "sosomt1.p"
  "{input no}"}
```
STD-0229 Use of lower-case and upper-case syntax in Progress Source Files

Summary
Use all lower case when writing Progress syntax.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0229</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style, Procedure Structure, Environment</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
All Progress syntax should be in lower case unless there is a specific standard which allows for mixed case or all upper case.

Rationale
Traditionally, most of the MFG/PRO source code has been written using lower-case syntax for all keywords.

Exceptions:
- Progress UI Builder generated code
- literal strings
- PROCEDURE and END PROCEDURE
- CASE and END CASE
- FUNCTION and END FUNCTION
- Anything beginning with an ampersand (&SCOPED-DEFINE, &GLOBAL-DEFINE, &IF, {&PREPROCESSOR}, etc.

The reason why we want to capitalize these syntax phrases is to make the sections of the code in which they occur stand out more for better readability.

Capitalization of Progress keywords should only be used to emphasize or bring attention to a particular section of code.

Examples

Right

Code

Example 1: "XYZ" is the literal string exception.
if pt_part begins "XYZ" then do:
end.

Example 2.
PROCEDURE checkSo_nbr:
    define variable check-so-nbr as character no-undo.
    return.
END PROCEDURE.

Wrong
Code

Example 1.
define variable C-So-Num as character no-undo.

For first so_mstr NO-LOCK NO-ERROR.
C-So-Num = so_nbr.

Example 2.
procedure checkSo_nbr:
  DEFINE VARIABLE check-so_nbr AS CHARACTER NO-UNDO.
  RETURN.
end procedure.

See Also

STD-0230 Always use END PROCEDURE with PROCEDURE, END FUNCTION with FUNCTION, and END CASE with CASE
STD-0230 Always use END PROCEDURE with PROCEDURE, END FUNCTION with FUNCTION, and END CASE with CASE

Summary
Always fully identify the type of end statement when using PROCEDURE, CASE, and FUNCTION.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0230</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style, Procedure Structure, Environment</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
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, END FUNCTION statements should always in be upper case.

Rationale
To improve code readability and to make these specific types of blocks stand out more clearly within the code.

Examples
Right

Code
```plaintext
PROCEDURE testProcedure:
  define input parameter choice as character no-undo.
  CASE choice:
    when "x" then do:
      end.
    when "y" then do:
      end.
    otherwise do:
      end...
  END CASE.
END PROCEDURE.
```

Wrong

Code
PROCEDURE testProcedure:
  define input parameter choice as character no-undo.
  CASE choice:
    when "x" then do:
    end.
    when "y" then do:
    end.
    otherwise do:
    end.
  end CASE.
END.

See Also

STD-0229 Use of lower-case and upper-case syntax in Progress Source Files
STD-0231 Don’t abbreviate ANY Progress syntax

Summary

Never abbreviate ANY Progress keywords or syntax.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0231</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Although the Progress language allows for abbreviations in statements, Progress syntax should not be abbreviated in MFG/PRO source files. Leaving out optional expressions and phrases are not a violation to this rule. For example, when invoking an internal procedure, it is not necessary to specify the “in this-procedure” option following the run statement.

Note, the standards checker does check for some violations of this rule such as ‘exclusive’ instead of ‘exclusive-lock’ or ‘as char’ instead of ‘as character’. However, the current standards checker does not check for all of them so do not rely on the standards checker for flagging all violations to this rule.

Rationale

The Progress language is constantly evolving and changing. It is, therefore, very important that the application software written in this language be robust to changes between each new release. Abbreviating Progress syntax based on the current Progress language conventions can lead to problems and ambiguities in the future. You cannot assume that currently accepted abbreviations for Progress keywords will be maintained in future releases nor can you assume that they will always expand out to the same keyword. Therefore, do not abbreviate Progress keywords in the code.

Examples

Right

Code

```progress
define variable customer_addr as character format "x(8)" no-undo.

for first cm_mstr exclusive-lock:
  if available cm_mstr then do:
    customer_addr = cm_addr.
    display customer_addr.
  end.
  ...
  ...
end.
```

Wrong

Code

```progress
define variable customer_addr as character format "x(8)" no-undo.

for first cm_mstr exclusive-lock:
  if available cm_mstr then do:
    customer_addr = cm_addr.
    display customer_addr.
  end.
  ...
  ...
end.
```
def var customer_addr as char format "x(8)" no-undo.

for first cm_mstr exclusive:
  if avail cm_mstr then do:
    customer_addr = cm_addr.
    disp customer_addr.
  end.
  ...
  ...
end.

See Also

N/A
STD-0232 Use of TABs

Summary

Do not use TABs within a source file as a means of indentation. All tabs should be converted to spaces.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0232</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>All</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Do not use TABs within a source file as a means of indentation. Tabs inside of strings should be avoided.

Rationale

The readability of a source file that has been formatted using tab stops can differ depending on the software editor used to display it. This can increase both the workload and the risk of introducing errors for developers who will be maintaining and enhancing the functionality contained within that source file. To ensure that the indentation of a source file is maintained across different software editors, the use of tab stops for indentation is not permitted.

Some editors (such as Visual Cafe and Ed for Windows) provide an option to convert a TAB key into spaces. Once this option is set, the editor will convert the TAB key to spaces, thus allowing the developer to use the TAB key since it is converted into spaces in the source code itself. If this option is not available in your editor, then the space bar must be used instead of the TAB key.

Please refer to STD-0233 for source file indentation rules.

Examples

Right

Code

```plaintext
for each wo_mstr no-lock:
    if available wo_mstr then do:
        end.
    else do:
        end.
end.
```

Wrong

Code
for each wo_mstr where wo_due_date < today no-lock:

<TAB> if available wo_mstr then do:

<TAB> end.
<TAB> else do:

<TAB> end.
end.

See Also

STD-0233 Source Code Indentation
STD-0233 Source Code Indentation

Summary

All source code should be indented appropriately to enhance readability of the code, and to make blocks of code more obvious.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0233</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>All</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Three spaces is suggested for each indentation level, as this is the indenting logic used in the automated “Strip & Beautify” program, which performs various code cleanup activities.

All end statements for each block of code should align with the statement starting the block.

Rationale

Properly indented code is easier to read and maintain.

Examples

Right

Code

EXAMPLE 1:

```define variable item_part as character format “x(18)” no-undo.
define variable item_desc1 as character format “x(24)” no-undo.
define variable item_loc as character format “x(8)” no-undo.
prompt-for pt_part.
for first pt_mstr where pt_part = input pt_part no-lock:
   assign
      item_part = pt_part
      item_desc1 = pt_desc1
      item_loc = pt_loc.
   display item_desc1 item_part item_loc.
end.
```

EXAMPLE 2:
for each ld_det
  where ld_part = part
  and ld_site = site
  and (ld_loc = loc or loc = "")
  and (ld_status = stat or stat = "")
  use-index ld_part_loc no-lock:

    if addr &lt;&gt; "" then do:
      for first locc_det
        fields (locc_site locc_loc locc_addr)
        where locc_site = site
        and locc_loc = ld_loc
        and locc_addr = addr
        no-lock:
        end. /* for first locc_det */
      if not available locc_det then next loadtt.
      ...
    end. /* if addr &lt;&gt; "" */
end. /* for each ld_det */

EXAMPLE 3:

PROCEDURE reservedLocationInventory:
  /* THE INTENT OF THIS PROCEDURE IS TO DETERMINE RESERVED */
  /* LOCATION INVENTORY ADJUSTMENTS NEEDED */
  define input parameter ip-atp-site like si_site no-undo.
  define input parameter ip-atp-part like pt_part no-undo.
  define input parameter ip-addr like ad_addr no-undo.
  define output parameter op-qty-all like in_qty_oh no-undo.
  define output parameter op-qty-oh like in_qty_oh no-undo.
  {gprun.i "sorlavla.p"
    "(input ip-atp-site,
      input ip-atp-part,
      input ip-addr,
      output op-qty-all,
      output op-qty-oh)"
  }
END PROCEDURE. /* resv-loc-atp */

Wrong

Code

EXAMPLE 1:

define variable item_part as character format "x(18)",
define variable item_desc1 as character format "x(24)",
define variable item_loc as character format "x(8)",
prompt-for pt_part.

for first pt_mstr where pt_part = input pt_part no-lock:
  assign
    item_part = pt_part
    item_desc1 = pt_desc1
    item_loc = pt_loc.
  display item_desc1 item_part item_loc.
end.

EXAMPLE 2:
loadtt:
for each ld_det
where ld_part = part
and ld_site = site
and (ld_loc = loc or loc = "")
and (ld_status = stat or stat = "")
use-index ld_part_loc no-lock:
if addr <> "" then do:
for first locc_det
fields (locc_site locc_loc locc_addr)
where locc_site = site
and locc_loc = ld_loc
and locc_addr = addr
no-lock:
end. /* for first locc_det */
if not available locc_det then next loadtt.
end. /* if addr <> "" */
end. /* for each ld_det */

EXAMPLE 3:

PROCEDURE reservedLocationInventory:
/* THE INTENT OF THIS PROCEDURE IS TO DETERMINE RESERVED */
/* LOCATION INVENTORY ADJUSTMENTS NEEDED */
define input parameter ip-atp-site like si_site no-undo.
define input parameter ip-atp-part like pt_part no-undo.
define input parameter ip-addr like ad_addr no-undo.
define output parameter op-qty-all like in_qty_oh no-undo.
define output parameter op-qty-oh like in_qty_oh no-undo.
{gprun.i "sorlavla.p"
"(input ip-atp-site,
 input ip-atp-part,
 input ip-addr,
 output op-qty-all,
 output op-qty-oh")"
END PROCEDURE. /* resv-loc-atp */
STD-0234 Always leave at least one space surrounding comparison and assignment operators.

Summary

Always leave at least one space before and after comparison operators (=, >, <, etc.). Also, leave at least one space before and after the assignment operator (=).

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0234</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>All</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Although the absence of spaces around comparison operators (=, >, <, >=, <=, <>) and the assignment operator (=) is permitted by Progress, the use of spaces improves the overall readability of the code.

Exceptions

- Tokens are strings parsed by other processors and are required to follow the specific format required by the external processor (V8:RunMode=Character,Web). Tokens must be coded exactly as specified and may not require spaces around assignment operators.

- Preprocessors and include file assignment statements do not need spaces surrounding the operators. This is because these statements are actually substitutions of code during compilation. For example:

```language
{mficgl02.i
 &gl-amount=tr_gl_amt
 &tran-type=tr_type
 &order-no=tr_nbr
 &dr-acct=trgl_dr_acct
 &dr-cc=trgl_dr_cc
 &drproj=trgl_dr_proj
 &cr-acct=trgl_cr_acct
 &cr-cc=trgl_cr_cc
 &crproj=trgl_cr_proj
 &entity=si_entity
 &find="false"
 &same-ref="icc_gl_sum"
}
```

Rationale

For better readability of the code, there should always be at least one space surrounding the comparison operators and the assignment operator.

/* Example 1. */

Example

Right
**Code**

```plaintext
for each so_mstr where so_due_date >= today no-lock:
   display so_nbr so_ship so_cust so_due_date.
end.
```

```plaintext
/* Example 2. */
create pt_mstr.
assign
   pt_part   = input pt_part
   pt_site   = input pt_site
   pt_loc    = input pt_loc
   pt_lot    = input pt_lot
   pt_status = input pt_status.
```

**Wrong**

**Code**

```plaintext
/* Example 1. */
for each so_mstr where so_due_date >= today no-lock:
   display so_nbr so_ship so_cust so_due_date.
end.
```

```plaintext
/* Example 2. */
create pt_mstr.
assign
   pt_part = input pt_part
   pt_site = input pt_site
   pt_loc  = input pt_loc
   pt_lot  = input pt_lot
   pt_status = input pt_status.
```

**See Also**

STD-0214 Allowed Comparison Operators
STD-0235 Proper usage of User Interface Tokens (RunMode)

Summary

The RunMode user interface token controls which user interface (UI) version (character, Windows or web) should include the source file. The RunMode UI token is a specific string in each source file to tell the ECO processing system and other utilities how to handle the file.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0235</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>MFG/PRO versions 8.5 and above</td>
</tr>
</tbody>
</table>

Description

Tokens are specific strings in each file to tell the ECO processing system and other utilities how to handle the file. The token must be correct for the file to be handled properly.

The RunMode user interface token controls which user interface versions (character, Windows or web) receive a file. The format of the user interface token is a comment that begins with V8:RunMode= followed by a comma-separated list (with no spaces) of the UI names (Character, Windows, Web) which require the source file.

The user interface token V8:RunMode=Web is no longer used in MFG/PRO versions eB and above – only V8:RunMode=Character and V8:RunMode=Windows are allowed. V8:RunMode=Character is used when the source file is to be excluded from Windows. V8:RunMode=Windows is used when the source file is to be excluded from CHUI, Net UI and Desktop.

A user interface token is required in source files that are not available to all UIs.

- Source files that are applicable to all UIs do not need a RunMode UI token. The absence of a RunMode UI token assumes the source file is applicable to all UIs.
- Tokens are always placed in a comment statement after the Revision History header section.
- Tokens should be on their own line.
- Token format is extremely important and must be followed exactly in order for the token scanner to read it correctly.
- The order of the RunMode list is important. Note that V8:RunMode=Character,Web is correct, V8:RunMode=Web,Character is incorrect. Use the tokens exactly as specified.

Never modify an existing RunMode UI token such that the program will no longer be supported in a particular UI without prior approval. For example, never un-web-enable a program. There needs to be an approved design specifying why the UI will no longer be supported.

- When modifying tokens according to the guidelines below, standard, "comment-out" coding will not work. Tokens can not be commented out because the tokens are already comments. Please follow the directions and actually remove the tokens when necessary. The token can be added to a new or existing file, but can not exist in a file with any other V8:RunMode token.
- Adding a token to a source file's header does not guarantee that the program will run correctly in the intended environment. Programmers must also include the proper GUI converter preprocessors and web enablement logic.
- The cktokens script will log an error if the source file is using the old style token for non-web-enabled
programs. It does not check the syntax of the token, nor validate whether or not it is required. The
cktokens script is invoked by the Standards Checker (ckstd) and during file check in and check out by the
ckin and ckout scripts. cktokens can also be run independently to validate tokens in a source file.

<table>
<thead>
<tr>
<th>Char</th>
<th>Web</th>
<th>Win</th>
<th>Token to use (pre-eB)</th>
<th>Token to use (eB and up)</th>
<th>Obsolete token previously indicate this UI restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>/*V8:RunMode=Character */</td>
<td>/*V8:RunMode=Character */</td>
<td>/*V8:ConvertMode=NoConvert */</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>/*V8:RunMode=Character,Web */</td>
<td>/*V8:RunMode=Character */</td>
<td></td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>No token required (DEFAULT)</td>
<td>No token required</td>
<td>(DEFAULT)</td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>/*V8:RunMode=Character,Windows */</td>
<td>No token required</td>
<td>/*V8:WebEnabled=No */</td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>/*V8:RunMode=Web */</td>
<td>/*V8:RunMode=Character */</td>
<td></td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>/*V8:RunMode=Web,Windows */</td>
<td>No token required</td>
<td></td>
</tr>
<tr>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>/*V8:RunMode=Windows */</td>
<td>/*V8:RunMode=Windows */</td>
<td></td>
</tr>
<tr>
<td>no</td>
<td>no</td>
<td>no</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

**Rationale**

A program may fail to operate in a UI version if the user interface token is not included, or properly specified.

**Examples**

**Right**

**Code**

Example of a file to be used in only a Window UI (Not enabled in the web UI):

```plaintext
/* sosomt.p - SALES ORDER MAINTENANCE */
/* Copyright 1986-2001 QAD Inc., Carpinteria, CA, USA. */
/* All rights reserved worldwide. This is an unpublished work. */
/* */
/* Revision: 1.0    BY: Joe Coder    DATE: 01/10/01  ECO: *YYYY* */
/* Revision: 1.3    BY: John Q Public  DATE: 03/19/01  ECO: *ZZZZ* */
/*-End of revision history----------------------------------------------*/
/* */
/*V8:ConvertMode=Maintenance */
/*V8:WebEnabled=No */
/*V8:RunMode=Windows */
```

**Wrong**

**Code**

Example 1: The UI token should be on its own line and in the specified order.
Example 2: Tokens can not exist in a file with any other V8:RunMode token and must include the proper GUI converter preprocessors. Tokens can not be commented out when no longer needed, because the tokens are already comments.

See Also

N/A
STD-0236 Proper usage of GUI Converter tokens (ConvertMode)

Summary

GUI Converter tokens are specific strings in each file to tell the GUI Converter how to handle the file. The GUI Converter token must be correct for the file to be converted properly.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0236</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities for MFG/PRO source code</td>
</tr>
</tbody>
</table>

Description

The ConvertMode token is used by the GUI Converter to modify the source file to run in Windows. (The GUI Converter is a standard QAD method for generating Windows versions of MFG/PRO character source code.) The source file generated will depend on the nature of the input file. For example, a maintenance source file requires different Windows handling than a report source file.

The format of the ConvertMode token is a comment that begins with `V8:ConvertMode=` followed by the desired output mode.

The ConvertMode token is always placed in a comment statement after the Revision History header section.

The ConvertMode token should be on its own line.

Adding the ConvertMode token to a source file's header does not guarantee that the program will run correctly in Windows. Programmers must also include the proper GUI Converter preprocessors as needed.

/*V8! ...*/ is used to enable code only for MFG/PRO for Windows. the GUI Converter removes the leading and trailing comments during the pre-pass and then converts the previously enclosed code during the output-generating pass.

```
Before convert:
message "Whoa!" /*V8! view-as alert-box */
After convert:
message "Whoa!" view-as alert-box.
```

/*V8-*/ ... /*V8+*/ is used to omit code from the converted version of the source file. The code found between these tokens is not processed by the GUI Converter and not written to the output file.

```
Before convert:
message "Smell the roses." /*V8-*/ pause. /*V8+*/
After convert:
message "Smell the roses." /*V8+*/
```

There are several tokens that can be embedded in the program to help the GUI Converter produce the desired output listed below. Only one ConvertMode tokens can exist in a file. Do not comment out an existing ConvertMode token to modify or delete it. If the token needs to be changed, the modification should be done on the existing ConvertMode token line.

Each of the GUI ConvertMode token options is discussed below.

1. `/*V8:ConvertMode=Maintenance*/`

This directive tells the GUI 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.

2. /*V8:ConvertMode=FullGUIReport*/ should be used for all new report programs. This directive tells the GUI 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 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".

3. /*V8:ConvertMode=Report*/ should be used for inquiry and report programs that are not yet compatible with the MFG/PRO FullGUIReport program template. This directive tells the GUI 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.

4. /*V8:ConvertMode=ReportAndMaintenance*/
This directive causes the GUI Converter to insert GUI logic which supports both a 'Maintenance Program' UI and a 'GUI' UI. 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. The appropriate attributes are activated at compile time based on preprocessor definitions inserted by the GUI Converter. In other words, if you have:

- a maintenance program: mymaint.p with a ConvertMode=Maintenance token - and -
- a report/inquiry program: myinquiry.p with a ConvertMode=Report - and -
- both of these programs call the same sub-program or use the same include file and that sub-program or include file has UI functionality -

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

5. /*V8:ConvertMode=NoConvert*/
When this ConvertMode token is found, the input file is copied to the target directory with a system "copy" command. This token is used for files that the GUI Converter does not need to modify to be Windows-compatible. Examples would be files with no user interface, or files that recognize the user interface without GUI Converter help, such as browses.

6. /*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.
See Also

STD-0162 What the GUI converter does to Full GUI reports
STD-0235 Proper usage of User Interface Tokens (RunMode)
STD-0238 Block Labels

Summary

General guidelines for labeling transaction blocks (i.e. blocks that use the keywords DO, FOR EACH, REPEAT, TRANSACTION).

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0238</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Labeling a block is useful for 2 purposes; handling the iteration of a block (next statement), and handling the error processing of a block (undo). If a developer feels the need to label a block (for, repeat, do), it is important that they follow these guidelines:

- The label should be in lower case (mainloop:)
- The label should describe the primary function or the action that is occurring within the block (mainloop:, delete-loop, sod_det-loop:, etc.)
- It is suggested, although not required, that all block labels be referenced later in the program.
- Although Progress allows for duplicate block labels within the same procedure, this practice is strongly discouraged. It is permissible for internal procedures to have block labels that are duplicated in a different internal procedure (e.g., mainloop).
- The label should be on a line by itself above the block header.
- Any NEXT, LEAVE, UNDO statement within the block should explicitly reference that label.
- Labeled blocks should have the block label as a comment next to the END statement for the block.

Rationale

Including block labels for large blocks helps the overall readability of the code. It is particularly useful for clearly identifying the error handling actions within nested blocks/loops.

Examples

Right

Code

In this example, the mainloop label is referred to numerous times and the next mainloop clearly identifies what block is affected.

```plaintext
mainloop:
repeat:
...
do transaction with frame a on endkey undo, leave mainloop:
...
if undo_del then next mainloop.
end.
end. /*mainloop*/
```

Wrong
Code

In this example, the developer labels 2 blocks (mainloop and transblock) on the same line as the code. Only one of the block labels is referenced in the code. Ideally, the END statement for REPEAT should have a comment noting it is for the mainloop block.

```qcode
mainloop: repeat:
    ...

    transblock: do transaction with frame a on endkey undo, leave:
        ...
        if undo_del then next transblock.
        end.

end.
```

See Also

N/A
STD-0240 Localization Tags

Summary

Localization tags is the standard method of embedding calls to localizations in MFG/PRO.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0240</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Localizations</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Localization Tags

By using pre-processor directives, "localization tags", can be dropped into MFG/PRO code that are not "turned on" in the MFG/PRO baseline code. The localization offices can insert these tags in the MFG/PRO baseline before a major release, and then by adding pre-processor global defines to a standard include file (post release), turn on the modifications/customizations/localizations for their customers.

The methodology is to insert pre-processor tags that are "empty" because they are undefined in standard product. The modified code must also include a standard include file, ecxustom.i, that is shipped empty. The localization office then edits that file, adding pre-processor global-define's for the customization they want to "turn on". It is hoped that most localizations can be accommodated with this scheme. Where it can't, the localization office will have to resort to invasive changes as before.

This standard describes:

- Standard form of a Localization Tag
- How to reserve a Localization Tag
- The localization control include file

Format of Localization Tag

The standard form of a localization tag is:

```
{&[filename]-Tag#}
```

in which:

- [filename] : Name of containing source file including the extension, where the dot is replaced by the ".-" sign (a dot caused problems in GUI)
- Tag : Standard part of every tag
  1. : Sequential number. Always starts with 1 for a given filename

an example would be:

```
{&SOSOMTA-P-TAG1}
```

Reserving a Localization Tag

From any corporate unix host (coli36, ohsu04, crsu04) execute the command:

tagdb

Select option: 1. Tag Maintenance
Enter the filename and the system will give next available tag. Use generated tags only. This will ensure other Localization sites dont use the same tag for different purposes.

**Sample Session**

coli36:92b /users/lrr $ tagdb

---

**Localization Tag Menu**

1. Tag Maintenance
2. Tag Report
3. Tag Browse
4. Quit

---

**Localization Tag Maintenance**

<table>
<thead>
<tr>
<th>Filename:</th>
<th>some.p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag Name:</td>
<td>SOME-P-TAG1</td>
</tr>
<tr>
<td>Description:</td>
<td>eB2</td>
</tr>
</tbody>
</table>

User: lrr
Date Created: 11/28/03
Time Created: 12:00:00

SOME-P-TAG1 is available and is reserved for a localization. Next time a users want to generate a tag for the same file SOME-P-TAG2 will be generated. Note that tags are release independent (ie once a tag has been reserved it should not be used in another release for different purposes)

**Localization Control Include File**

The following include file will be shipped with the MFG/PRO baseline release and must be included at the top of any program with localization tags:

```c
/* cxcustom.i - The localization control file */
/* standard copyright goes here */
/* This include file is used to control the "turning on" of localization. By default this file is shipped empty so that no localization are turned on. The localization offices turns on the tags with pre-processor directives like:
   &GLOBAL-DEFINE SOSOMTA-P-TAG1 ~(cx0001.i~)
   They do this by editing this file as a post release activity and before installing "localized" code at the client site.
   Please note that this technique may not be supported in future releases.
*/
```

This localization control include file will be added to the MFG/PRO source file in the following format:

```c
(cxcustom.i filename)
```

in which:

filename : Name of containing source file including the extension, where the dot is replaced by the "-" sign. This filename allows for special structuring of the tags in case the amount of tags gets very large. Although there is a preprocessor directive, &FILE-NAME, which evaluates to the file currently being compiled, this preprocessor give difficulties in actual coding. The filename can be used, together with other preprocessor functions, to build the name of an include file that is to be used inside of cxcustom.i. That include file inside cxcustom.i can
QAD Development Standards

contain the global preprocessor directives for the file currently being compiled. While this is not necessary in the short run, it's good to know that there is potentially a way to chunk up (reduce the number of entries of) the cxcustom.i file if required.

Rationale

The localization tag methodology allows the enhancement of MFG/PRO without having the drawbacks of having conditional checks of customized code at runtime, vastly complicating the job of testing and validating the MFG/PRO code baseline. It is anticipated that API's will become much more prevalent as the product moves forward and therefore the preferred method for the integration of enhancements. Since the API's are not generally available, this methodology will serve as a bridging technique until the API's are generally available.

The advantages of this methodology are:

- Can be used where API's are not yet available.
- Separates standard MFG/PRO source code from localization and add-on source code enhancements.
- Makes it easier to upgrade the localizations and add-on's to a new version of MFG/PRO.
- Allows the localizations and add-ons to be version independent.
- Makes it easier to mix and match various localizations and add-ons.

The disadvantages include:

- Allows for high coupling of customizations with the baseline source, a generally undesirable trait for software engineering. The preferred method is to use an API thus having the contract explicitly specified between the standard code and localized code.
- There is no guarantee that the tags will remain in the correct place as the code is moved forward by R&D.
- Discipline has to be exercised by the various localization offices using the localization tag database to make sure they don't stomp on each other's tags.
- Doesn't meet all localization requirements so some invasive code modifications will probably still have to be done. These invasive code modifications fall outside the scope of this standard.

Examples

Right

Design

Poland wants to add localization tags to a section of "some.p" that does tax calculations:

Code
Wrong

Design

Same requirement with incorrect implementation. Notes in blue.

Code

```/* some .p - Some mfg/pro program */

Header stuff....

cxcustom.i some-p

/* DISCOUNT TABLE VALIDATION */
/* ADDED TWO ARGUMENTS &DISP-MSG AND &WARNING */

{addsclst.i &disc-list = "cm_mstr.cm_pr_list"
 &curr = "cm_mstr.cm_curr"
 &disc-list-req = "no"
 &undo-label = "setb2"
 &with-frame = "with frame b2"
 &disp-msg = "yes"
 &warning = "yes" }

if {txnew.i} or {txload.i} then do:
undo_adcsmtc = true.

(&amp;SOME-P-TAG1)

(gprun.i "adcsmtc.p")
if undo_adcsmtc then
undo setb2, retry.
end. /* if {txnew.i} or {txload.i} then do */
```
/* some .p - Some mfg/pro program */

Header stuff....

/* {cxcustom.i} */

missing {cxcustom.i} include file call here. Program will still compile but customizations can't be easily turned on as a post release activity. missing filename (some-p): format should be {cxcustom.i some-p}.

/* DISCOUNT TABLE VALIDATION */
/* ADDED TWO ARGUMENTS &DISP-MSG AND &WARNING */

{addsclst.i &disc-list = "cm_mstr.cm_pr_list"
 &curr = "cm_mstr.cm_curr"
 &disc-list-req = "no"
 &undo-label = "setb2"
 &with-frame = "with frame b2"
 &disp-msg = "yes"
 &warning = "yes" }

if {txnew.i} or {txload.i} then do:
  undo_adcsmtc = true.

{&SOME-TAG1}

Doesn't obey the naming standard. Should be {&SOME-P-TAG1}.
Didn't use the localization tag db to generate the tag, another localization group may be using SOME-P-TAG1

{gprun.i "adcsmtc.p"
  if undo_adcsmtc then
    undo setb2, retry.
end. /* if {txnew.i} or {txload.i} then do: */

See Also

N/A
STD-0241 Java File Suffixes

Summary

Standard file suffixes must be used for various types of Java files.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0241</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Java</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Java Programs</td>
</tr>
<tr>
<td>Applicability</td>
<td>All Java development activities</td>
</tr>
</tbody>
</table>

Description

The following standard Java file suffixes should be used for the corresponding types of files.

<table>
<thead>
<tr>
<th>File Type</th>
<th>Suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java source</td>
<td>.java</td>
</tr>
<tr>
<td>Java byte-code</td>
<td>.class</td>
</tr>
<tr>
<td>Java archive</td>
<td>.jar</td>
</tr>
</tbody>
</table>

The Java language requires that the standard file suffixes are used for various types of Java files. Any other file suffixes will cause either compile time or run-time errors.

Examples

Right

Code

```
ApplicationPresentation.java
ApplicationPresentation.class
jbase.jar
```

Wrong

Code

```
ApplicationPresentation.javax
ApplicationPresentation.classx
jbase.jarx
```

See Also

N/A
STD-0242 Java Package Name

Summary

All Java packages produced by QAD must start with 'com.qad.'

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0242</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Java</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Java Programs</td>
</tr>
<tr>
<td>Applicability</td>
<td>All Java development activities</td>
</tr>
</tbody>
</table>

Description

Java allows you to group classes in a collection called a package. Packages are convenient for organizing Java files and for separating your code from code libraries provided by third party vendors. To avoid conflict with package names provided by others, Sun Microsystems recommends that a company's domain name in reverse order is used as a package prefix. To make the packages produced by QAD distinct, all QAD-produced packages must start with 'com.qad'. Java package names must be single lowercase words separated by periods (.).

When you make a package it is your responsibility to place the object files in the correct subdirectory. For example, if you compile a file that starts with the line

    com.qad.util;

then, you must put the resulting class file into the subdirectory com/qad/util.

Rationale

A Java package prefix helps to organize Java files and keep them distinct from packages provided by third-party vendors.

Examples

Right

Code

```
package com.qad.base.pres;
package com.qad.base.data;
package com.qad.base.proc;
```

Wrong

Code

```
package com.qad.base.pres;
package qad.base.pres;
package base.pres;
 package proc;
```
STD-0243 Naming Guidelines for Java Classes, Interfaces, Methods, Variables and Parameters

Summary

Use the following guidelines for naming Java classes, interfaces, methods, variables, static variables (constants) and parameters.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0243</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Java</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Java Programs, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All Java development activities</td>
</tr>
</tbody>
</table>

Description

1. Classes
Class names should be nouns, in mixed case with the first letter of each internal word including the first word is capitalized. Try to keep your class names simple and descriptive. The choice of a name should be mnemonic - that is, designed to indicate to the casual observer the intent of its use. Use whole words - avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML) - this general guideline is also applicable for the following naming guidelines. Underscore (_) and dollar sign ($) should not be used.

2. Interfaces
In Java, interfaces are a mechanism to implement multiple inheritance. Interface names should be nouns similar to class names, in mixed case with the first letter of each internal word including the first word is capitalized. Try to keep your interface names simple and descriptive. It should start with 'I' (capital I for Interface). Underscore (_) and dollar sign ($) should not be used.

3. Methods
Methods should start with verbs, in mixed case with the first letter of the first word in lowercase and the first letter of each internal word capitalized. Getters (methods that 'get' attributes) and setters (methods that 'set' attributes) should begin with 'get' / 'set' verb and return the appropriate object type. Boolean getters should use 'is' or 'can' as a prefix, such as 'isUndoable' rather than 'getUndoable'. Underscore (_) and dollar sign ($) should not be used.

4. Instance Variable
A variable that holds the instance of an object is called instance variable. In naming instances variables, we distinguish between objects created by a class and objects just referenced within the class. Variables that reference objects that the class is responsible for creating must begin with my<Object> whereas those that reference that other classes are responsible for creating must begin with the<Object>. The name of <Object> follows the same rules as the class name. An instance variable must be made private unless there is a good reason like performance etc. Underscore (_) and dollar sign ($) should not be used.

5. Class variable
A variable whose scope is the entire class and declared outside of any method is called a class variable. The variable names should be in mixed case with the first letter of the first word in lowercase and the first letter of each internal word capitalized. Variable names should be short yet meaningful. One or two character variable names should be avoided. A class variable should not start with my, the or p so that they will not be confused with instance variables or parameters. An class variable must be made private unless there is a good reason like performance etc. Underscore (_) and dollar sign ($) should not be used.

6. Local variable
A variable local to a method is called a local variable. Similar to a class variable, the variables should be in mixed case with the first letter of the first word in lowercase and the first letter of each internal word capitalized. Variable names should be short yet meaningful. One or two character variable names should be avoided. A local variable should not start with my, the or p so that they will not be confused with instance variables or parameters. Underscore (_) and dollar sign ($) should not be used.

7. Static variable (constant)
In Java, you use the keywords final static before a variable to denote a constant. You can assign to the variable only once, and then its value is set once and for all. The names of the constants should be all capitals with words separated by underscores. This is also applicable for a group of constants like constants for different data types. Numerical constants (also called literals) should not be coded directly, except for -1, 0, and 1, which can appear in a for loop as counter values. Instead, they should be declared as constants. Likewise string and character literals should be only found in constant declarations.

8. Method parameter
A method parameter name should start with lower case 'p' (for 'parameter'). Similar to a class name, in a multi-word method parameter name, the first letter of each word is capitalized. Underscore (_) and dollar sign ($) should not be used.

**Rationale**

Following the guidelines for naming Java classes, interfaces, methods, variables and parameters make the code easy to read, understand and maintain.

**Examples**

**Right**

**Design**

**Code**

1. Classes

```java
class Model;
class Images;
class PresentationManager;
```

2. Interfaces

```java
interface IBooleanListener;
interface IEditable;
```

3. Methods

```java
run();
getSalesTax();
getBackground();
isFullTime();
```

4. Instance variable

```java
I18NManager theI18NManager;
```

5. Class variable

```java
static double socialSecurityRate = 7.2;
```

6. Local variable

```java
int anApple;
float aWidth;
boolean customerType;
String itemStatus;
```

7. Static variable (constant)

```java
final static int MIN_RECS = 4;
final static int DATATYPE_REAL = 5;  //DATATYPE denotes a group
final static int DATATYPE_INT = 6;
final static int DATATYPE_CHAR = 7;
```
8. Method parameter

<table>
<thead>
<tr>
<th>pValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSalesTax</td>
</tr>
</tbody>
</table>

Wrong

Design

Code

1. Class

```java
class model;
class images;
class presentationManager;
class Presentation_Manager;
class $PresentationManager;
```

2. Interfaces

```java
interface Boolean_Listener;
interface Editable;
interface editable;
```

3. Methods

```java
Run();
get_SalesTax();
Background();
getFullTime();  //FullTime is an boolean
```

4. Instance variable

```java
I18NManager aI18Nmanager;
```

5. Class variable

```java
static double theSocialSecurityRate = 7.2;
static double theSocial_Security_Rate = 7.2;
```

6. Local variable

```java
int Apple;
float myWidth;
boolean customer_Type;
String ItemStatus;
```

7. Static variable (constant)

```java
final static int Min_Recs = 4;
final static int TIPEReal = 5;
final static int Type_Int = 6;
```

8. Method parameter

```java
Value
Pvalue
SalesTax
Sales_Tax
```

See Also

N/A
STD-0244 Java Source Code Comments

Summary

Follow the guidelines described in the Detailed Description section on using comments in Java code.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0244</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Java</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Java Programs, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All Java development activities</td>
</tr>
</tbody>
</table>

Description

Java Comments
Java programs can have two kinds of comments: implementation comments and documentation comments. Implementation comments are like those found in C++, which are delimited by /*/ and //. Documentation comments (known as “doc comments”) are Java-only, and are delimited by /*/-. Doc comments can be extracted to HTML files using the Javadoc tool.

- Implementation comments are for comments about the particular implementation. Documentation comments are meant to describe the specification of the code, from an implementation-free perspective, to be read by developers who might not necessarily have the source code at hand.

- Comments should be used to give overviews of code and provide additional information that is not readily available in the code itself. Comments should contain only information that is relevant to reading and understanding the program. For example, information about how the corresponding package is built or in what directory it resides should not be included as a comment.

- Discussion of nontrivial or non-obvious design decisions is appropriate, but avoid duplicating information that is present in (and clear from) the code. It is too easy for redundant comments to get out of date. In general, avoid any comments that are likely to get out of date as the code evolves.

- Do not leave commented-out code. If there is any code commented out, delete it.

- The frequency of comments sometimes reflects poor quality of code. When you feel compelled to add a comment, consider rewriting the code to make it clearer.

I. Implementation Comments
Programs can have three styles of implementation comments: block, single-line and trailing. Use them where they are appropriate in the source code.

1. Block Comment

- Block comments are used to provide descriptions of the implementation. Block comments should be used at the beginning of each file (see development standard STD-0196 'Source File Formatting - Header' for more information) and before each method. They can also be used in other places, such as within methods. Block comments inside a method should be indented to the same level as the code they describe.

- A block comment should be preceded by a blank line to set it apart from the rest of the code.

```java
/*
 * Here is a block comment.
 */
```
2. Single-Line Comment
Short comments can appear on a single line indented to the level of the code that follows. If a comment can't be written in a single line, it should follow the block comment format. A single-line comment should be preceded by a blank line. Here's an example of a single-line comment in Java code:

```java
if (condition)
{
    statements;
    // Handle the condition.
    statements ...
    ...
}
```

3. Trailing Comment
Very short comments can appear on the same line as the code they describe, but should be shifted far enough to separate them from the statements. If more than one short comment appears in a chunk of code, they should all be indented to the same tab setting. Avoid the assembly language style of commenting every line of executable code with a trailing comment. Here's an example of a trailing comment in Java code:

```java
if (a == 2)
{
    return true; //special case
}
else
{
    return isPrime(a); //works only for odd a
}
```

Note:
Implementation comments should not be enclosed in large boxes drawn with asterisks or other characters. Comments should never include special characters such as form-feed and backspace. An exception is the file header described in development standard STD-0246 Java Source File Organization and STD-0196 Source file formatting - Header (pre-eB2).

II. Documentation Comments
Doc comments (also called Javadoc comments) describe Java classes, interfaces, constructors, methods and variables. Each doc comment is set inside the comment delimiters /*...*/, with one comment per public class, method, interface or constructor. A doc comment is made up of two parts – a description followed by zero or more tags, with a blank line (containing a single asterisk *) between these two sections. Every class, interface, constructor and method definition must have a doc comment associated with it.

Use Javadoc macros
- @param to describe parameters of method,
- @return for return value,
- @exception if the method throws any exception and
- @see # (where applicable) to provide link to a related method(s) or variable(s). This comment should appear just before the declaration:

```java
/**
 * This is the description part of a doc comment
 * @tag Comment for the tag
 */
```

- The first line is indented to line up with the code below the comment, and starts with the begin-comment symbol (/**) followed by a return.
- Subsequent lines start with an asterisk *. They are indented an additional space so the asterisks line up. A space separates the asterisk from the descriptive text or tag that follows it.
- The last line begins with the end-comment symbol (*) indented so the asterisks line up and followed by a return. Note that the end-comment symbol contains only a single asterisk (*).
- If you need to give information about a class, interface, variable, or method that isn't appropriate for documentation, use an implementation.
block comment or single-line comment immediately after the declaration. For example, details about the implementation of a class should go in such an implementation block comment following the class statement, not in the class doc comment.

- Javadoc comments should not be positioned inside a method or constructor definition block, because Java associates documentation comments with the first declaration after the comment.

- All public classes, variables, methods must be commented for Javadoc use.

- For re-implemented methods, Javadoc will pick up the Javadoc information of the super class method. Therefore, Javadoc comment for the re-implemented method in the sub-class need not be entered if the Javadoc comment of the super-class method is applicable to the re-implemented method. However, if the purpose/use of the re-implemented method is different, then it should be documented for Javadoc.

- All interface methods must be documented for Javadoc. If a class is implementing an interface then it is sufficient to use the @see tag to document the implemented method and it provides a link to the relevant method. Again, if the implementation of the method is different from the intended interface it should be documented.

- Comments not intended for Javadoc use should never contain /**..*/.

- Provide descriptive information about the class elements instead of merely repeating its name in sentence form. For example, if method description uses only the words that appear in the method name, then it is adding nothing at all to what you could infer. The ideal comment goes beyond those words and should always reward you with some bit of information that was not immediately obvious from the API name. See ‘Code’ section for an example. Exceptions to the guideline are the getter and setter methods (the method that ‘gets’ or ‘sets’ an attribute).

**Rationale**

Following the guidelines on comments helps to improve readability and maintainability of source code. It is a Java language requirement to use Javadoc tags to generate HTML help documentation of Java Source code.

**Examples**

**Right**

**Code**

I. Implementation Comments
1. Block Comment

```java
/* Return an IOptionHolder for the 'close' action */
private IOptionHolder getCloseOptionHolder()
```

2. Single-Line Comment

```java
// This method is used by help
public Widget getCurrentWidget()
```

3. Trailing Comment

```java
Vector myInMsgs = new Vector(); // In-bound messages.
```

II. Documentation Comments
... /** The Presentation class supplies all generic behavior for the top level presentation object. It can be sub-classed to supply application behavior-dependent. */ @author Robert Webster @version 20-May-1999 */

public class Presentation extends GroupPanelWidget implements IPresentation, ILookupService, IXmlizable {
   /* Constructor(s) */

   /**
    * Constructor
    */
   public Presentation() {
      super();
      myProcessName = new Name("");
   }

   /* Public Method(s) */

   /**
    * Get an Icon
    * @param pIcon - the name of icon to be returned.
    * @return The ImageIcon for the required Icon.
    */
   public ImageIcon getIcon(String pIcon)
   {
      ...}

   // Descriptive use of Documentation Comment:
   // This description more completely defines what a tool tip is, in the larger context of registering and being displayed in response to the cursor.
   /**
    * Registers the text to display in a tool tip. The text displays when the cursor lingers over the component.
    * @param text The string to display. If the text is null, the tool tip is turned off for this component.
    */
   public void registerToolTipText(String text)
   {
      ...
   }

   // II. Documentation Comments
   // The following example of Javadoc is wrong because the Javadoc comments appear inside a method body.
   ... /* **
    * Registers the text to display in a tool tip. The text displays when the cursor lingers over the component.
    * @param text The string to display. If the text is null, the tool tip is turned off for this component.
    */
   public void registerToolTipText(String text)
   {
      ...}
Non-descriptive use of Documentation Comment:
The description below says nothing beyond what you know from reading the method name. The words "set", "tool", "tip", and "text" are simply repeated in a sentence.

```java
/**
 * Registers the tool tip text.
 * @param text The text of the tool tip.
 */
public void registerToolTipText(String text)
{
}
```

See Also

STD-0196 Source file formatting - Header (pre-eB2
STD-0246 Java Source File Organization
STD-0245 Using Java Control Statements

Summary

Follow the guidelines described in the Detailed Description section for using Java control statements.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0245</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Java</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Java Programs, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All Java development activities</td>
</tr>
</tbody>
</table>

Description

1. if - else Statement

Place the if keyword and conditional expression on the same line. The else if (condition) and else statements should appear on separate lines and must be aligned with the if statement as shown below.

Syntax:

```java
if (expression)
{
    statement;
}
else if (expression)
{
    statement;
}
else
{
    statement;
}
```

Braces should be used even if the body of the statement has only one line.

```java
if (expression)
    statement;    //AVOID! THIS OMITS THE BRACES {}!
```

2. for Statement

A for statement should have the following form:

```java
for (initialization; condition; update)
{
    statements;
}
```

An empty for statement (one in which all the work is done in the initialization, condition, and update clauses) should have the following form:

```java
for (initialization; condition; update);
```

When using the comma operator in the initialization or update clause of a for statement, avoid the complexity of using more than three variables. If needed, use separate statements before the for loop (for the initialization clause) or at the end of the loop (for the update clause).

3. while Statement

The while construct uses the same layout format as the if construct. The while keyword must appear on its own
line, immediately followed by the conditional expression. The statement block is placed on the next line.

Syntax:

```java
while (expression)
{
    statement;
}
```

4. do-while Statement
The do-while form of the while construct should appear as shown below:

Syntax:

```java
do
{
    statement;
} while (expression);
```

5. switch Statement
The switch construct uses the same layout format as the if construct. The switch keyword should appear on its own line, immediately followed by its test expression. The statement block is placed on the next line.

Syntax:

```java
switch (expression)
{
    case n:
        statement;
        break;
    case x:
        statement;
        //continue to default case
    default:
        statement;
        break;
}
```

Note:
The switch (expression) statement should appear on a separate line of its own. Each case statement within the switch block must appear on a separate line, properly indented (see standard STD-0233 ‘Source Code Indentation’ for more information on indenting) from the switch statement. Similarly, the statements within each case block must be indented as shown in the Syntax above. Each case statement block must have either a ‘break’ statement or a single line comment to indicate the control ‘continues’ (see the syntax above). Every switch statement should include a default case. The break in the default case is redundant, but it prevents a fall-through error if later another case is added.

6. try/catch/finally Statement
The try-catch construct is similar to the other statements. try keyword should appear on its own line followed by the open brace (optionally on the same line) followed by the statement body followed by the close brace on its own line. Any number of catch phrases are next consisting of catch keyword and the exception expression on its own line followed by the catch body followed by the close brace on its own line. The finally clause is the same as a catch.

Syntax:

```java
try
{
    statement;
}
catch (ExceptionClass e)
{
    statement;
}
finally
{
    statement;
}
```
A try-catch statement may also be followed by finally, which executes regardless of whether or not the try block has completed successfully.

**Rationale**

Following the guidelines on using Java control statements helps to improve readability and maintainability of source code.

**Examples**

**Right**

**Code**

1. **if - else Statement**

```java
if (pName.equals(STR_Severity))
{
    aRetVal = Integer.toString(mySeverity);
}
else if (pName.equals(STR_Messages))
{
    aRetVal = myMessageDataSet;
}
else
{
    throw new UnknownItemException(getName(), pName);
}
```

2. **for Statement**

```java
for (int i = 0; i < aDescs.length; i++)
{
    myDescSet.put(aDescs[i].getName(), aDescs[i]);
}
```

3. **while Statement**

```java
while (lvl < exp.myLevel)
{
    buf.append("\n");
    lvl++;
}
```

4. **do-while Statement**

```java
do
{
    writeProperty(aVar, (Property) anEnum.nextElement(), pLevel + 1);
} while (anEnum.hasMoreElements());
```

5. **switch Statement**
```java
switch (pType) {
    case DataDtd.IDENT_Name:
        myType.setName((String) pValue);
        break;
    case DataDtd.IDENT_Lockable:
        myType.setLockable(((Boolean) pValue).booleanValue());
        break;
    case DataDtd.IDENT_Flushable:
        myType.setFlushable(((Boolean) pValue).booleanValue());
        break;
    case DataDtd.IDENT_KeyFromService:
        myType.setKeyFromService(((Boolean) pValue).booleanValue());
        break;
    default:
        invalidAttr(pType);
        break;
}
```

6. try/catch/finally Statement

```java
try {
    setPriority(MIN_PRIORITY);
    unlock();
    yield();
    go();
    lock();
} catch (Exception e) {
    e.printStackTrace();
} finally {
    remove();
    Trace.trace(ModelThread.class,"stop " + Funcs.shortName(this)+");
}
```

Wrong Code

See Also

STD-0233 Source Code Indentation
STD-0246 Java Source File Organization

Summary

The Java source file should have the ordering for various code sections described in the Detailed Description.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0246</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Java</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Java Programs,Program Style,Procedure Structure</td>
</tr>
<tr>
<td>Applicability</td>
<td>All Java development activities</td>
</tr>
</tbody>
</table>

Description

1. Source file header
   A Java source file must contain a file header as described in ‘Source File Formatting - Header’ Development Standard STD-0196. The file header contains the file name, copyright statement, author, revision tag etc. See the section ‘Detailed Description and Revision History Formats’ in the development standard for Java source files.

2. Package and Import Statements
   The first non-comment line of a Java source file is a package statement, if it is applicable. An exception to the rule is a stand alone class like QADApplet.java which does not belong to a specific package. After that, import statements can follow. Package and Import statements beginning with the java.* imports, then third party's and finally QAD’s;

   When importing, specify down to class level, unless you’re using most of the classes in a package. For example:

   ...the import statement must look like this if you are importing a date package:

   ```java
   import java.util.date;
   ```

   ...instead of importing every package under java.util with the following statement:

   ```java
   import java.util.*;  //wrong!
   ```

3. Class/Interface Components
   a) Each Java source file contains an interface or a single public class with optionally one or more package level protected or private classes. The following table describes the parts of a class or interface declaration in the order that they must appear.

<table>
<thead>
<tr>
<th>Declaration Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class/Interface documentation comment</td>
</tr>
<tr>
<td>Class/Interface statement</td>
</tr>
<tr>
<td>Class constants</td>
</tr>
<tr>
<td>Class (static) variables</td>
</tr>
<tr>
<td>Instance variables</td>
</tr>
<tr>
<td>Constructors</td>
</tr>
<tr>
<td>Methods</td>
</tr>
</tbody>
</table>

   b) Within each type of class component, the elements must be sequenced by their access type i.e. public, protected, package and private.
c) Within methods, the class method should appear first followed by instance methods. The instance methods must be sequenced based on their type.

<table>
<thead>
<tr>
<th>Method Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
</tr>
<tr>
<td>Re-implementations of base class methods</td>
</tr>
<tr>
<td>Implementations of interfaces</td>
</tr>
</tbody>
</table>

**Rationale**

Organizing file header, package, import statements and class, interface declarations and methods provides consistent placement of them in the source file and helps readability and maintainability of source code.

1. **Source file header**
   See STD-0196 Source file formatting - Header (pre-eB2).

2. **Package and Import Statements**

   ```java
   package com.qad.base.pres;
   import java.awt.Container;
   import java.awt.Cursor;
   import java.util.Hashtable;
   import com.qad.base.util.INamespace;
   import com.qad.base.util.NameSpaceException;
   import com.qad.base.util.NameBindingException;
   import com.qad.base.util.TimingLogger;
   import com.qad.base.proc.IProcess;
   ```

3. **Class and Interface Declarations**
See Also

STD-0196 Source file formatting - Header (pre-eB2)
STD-0247 Java Source Code Formatting

Summary

Use the guidelines described in the Detailed Description section for formatting Java source code

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0247</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Java</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Java Programs, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All Java development activities</td>
</tr>
</tbody>
</table>

Description

1. Using TAB in source code
Do not use tab character as a means of indentation within a source file. For consistency and better readability, developers should use 3 spaces indentation for all source code. Set TAB in your editor to 3 spaces. All code segment that follows class, method, control statements etc should be indented. Refer to Development Standard STD-0232 ‘Use of TABs’ and STD-0233 ‘Source Code Indentation’ for more information on this topic.

2. Line length
A line length should have 80 characters or less because lines greater than 80 characters are not handled well by many terminals and tools. Refer to Development Standard STD-0227 ‘Maximum Acceptable Source Line Length’ for more information on this topic.

3. Blank Line
Blank lines improve readability by setting off sections of code that are logically related.

Two blank lines should always be used in the following circumstances:
- Between sections of a source file as listed in Class/Interface Components in Development Standard on "Java Source File Organization".
- Between class and interface definitions.

One blank line should always be used in the following circumstances:
- Between methods
- Between the local variables in a method and its first statement
- Before a block or single-line comment
- Between logical sections inside a method to improve readability

4. Blank Spaces
Blank spaces should be used in the following circumstances: A blank space should appear after commas in argument lists. All binary operators should be separated from their operands by spaces. Blank spaces should never separate unary operators such as unary minus, increment ("++"), and decrement ("--") from their operands. The expressions in a ‘for’ statement should be separated by blank spaces.

5. File length
Files longer than 2000 lines are cumbersome and should be avoided.

6. Wrapping lines
When an expression will not fit on a single line, break it according to these general principles:
- Break after a comma.
- Break before an operator.
- Prefer higher-level breaks to lower-level breaks.
- Indent one indentation unit (3 spaces) for a new line of a long expression.
- Line wrapping for if statements should generally use the two indentation units rule (6 spaces), since conventional (one indentation unit) indentation makes seeing the body difficult. If the line is squished against the right margin, use only one indent instead.

7. Braces
The starting brace must be on a separate line. The ending brace must be on a separate line and aligned with the conditional or the starting brace.
8. Parentheses

It is generally a good idea to use parentheses liberally in expressions involving mixed operators to avoid operator precedence problems. Even if the operator precedence seems clear to you, it might not be to others—you shouldn’t assume that other programmers know precedence as well as you do.

Rationale

Following the guidelines on Java source code formatting helps to improve readability and maintainability of source code.

Examples

Right

Code

3. Blank Line
package com.qad.base.mgr;

import java.rmi.RemoteException;
import com.qad.base.util.NameBindingException;
import com.qad.base.util.NameSpaceException;
import com.qad.base.util.FailureException;
import com.qad.base.util.NoSuchItemException;

/**
 * This is the root manager for all Project X applications. It contains
 * common startup functionality to support:
 * Local loading of managers .txt file (to be deprecated).
 * Local resource loading of managers .xml file.
 * Remote loading of managers .xml file.
 *
 * @author Brett Bennett
 * @version 24-March-2000
 */
public class ApplicationManager extends RootManager {
  /* Public Constructor(s) */
  /**
   * Construct an application manager using the given environment
   * manager.
   * If the Managers parameter has an extention load it directly,
   * otherwise
   * use a file resource manager to load it. If the RMIHost parameter is
   * given then use a client root manager to fetch it.
   * @param pEnvMgr an enviroment manager.
   */
  public ApplicationManager(IEnvironmentManager pEnvMgr)
      throws FailureException, RemoteException {
    this.addManager(pEnvMgr);
    RootManager aManager = null;
    String managersFile = "Managers.txt";
    String resourceDir = "C:/Run";
    boolean isRemote = false;

    try {
      String mgrs = pEnvMgr.getInfo(IEnvironmentManager.INFO_Managers);
      if(mgrs != null)
        { managersFile = mgrs; }
    }
    catch(NoSuchItemException e) {
    }
  }
}

4. Blank Spaces
a = a + c + d;
a = (a + b) / (c * d);
...
for (expr1; expr2; expr3)

6. Wrapping lines
- Examples of breaking method calls using one indentation unit (3 spaces):

```java
method(longExpression1, longExpression2, longExpression3,
       longExpression4, longExpression5);

variable = method1(longExpression1,
                    method2(longExpression2,longExpression3));
```

- Following is an example of breaking an arithmetic expression. Here, the break occurs outside the parenthesized expression, which is at a higher level.

```java
longName1 = longName2 * (longName3 + longName4 - longName5) + 4 * longName6;
```

- Following is an example of indenting method declaration. Use one indentation units (3 spaces).

```java
someMethod(int anArg, Object anotherArg, String yetAnotherArg,
            Object andStillAnother)
{
  ...
}
```

- Line wrapping for if statements should generally use the two indentation units rule (6 spaces), since conventional (one indentation unit - 3 spaces) indentation makes seeing the body difficult:

```java
if ((condition1 && condition2)
    || (condition3 && condition4)
    ||!(condition5 && condition6))
{
  doSomethingAboutIt();
}
```

//OR USE THIS
```java
if ((condition1 && condition2) || (condition3 && condition4)
    ||!(condition5 && condition6))
{
  doSomethingAboutIt();
}
```

7. Braces
```java
public ManagerReader(String pFile)
    throws IOException, XmlException
{
  super(new ManagerDtd());
  this.setReader(getReaderForResource(pFile, false));
}
```

```java
public ManagerReader(Reader pReader, ManagerDtd pDtd)
{
  super(pReader, pDtd);
}
```

8. Parentheses:
```java
if ((a == b) && (c == d))
Wrong
```
Code

4. Blank Spaces

```java
a=a + c + d;
a=(a+b)/(c*d);
...
for(expr1:expr2:expr3)
```

6. Wrapping lines

```java
longName1 = longName2 * (longName3 + longName4
- longName5) + 4* longName6;

if ((condition1 && condition2)
|| (condition3 && condition4)
||!(condition5 && condition6)) {    //BAD WRAPS
    doSomethingAboutIt();    //MAKE THIS LINE EASY TO MISS
}
```

7. Braces

```java
public ManagerReader(String pFile)
    throws IOException, XmlException {    //Brace should not start here
    super(new ManagerDtd());
    this.setReader(getReaderForResource(pFile, false));
}

public ManagerReader(Reader pReader, ManagerDtd pDtd) {    //Brace should not start here
    super(pReader, pDtd);
}
```

8. Parentheses:

```java
if (a == b && c == d)
```

See Also

STD-0232 Use of TABs
STD-0233 Source Code Indentation
STD-0227 Maximum acceptable Source Line Length - 80 characters
STD-0246 Java Source File Organization
STD-0248 Variable Declaration and Initialization in Java

Summary

Follow the guidelines described in the Detailed Description section on declaration and initialization of variables in Java.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0248</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Java</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Java Programs, Program Style, Variable Definition</td>
</tr>
<tr>
<td>Applicability</td>
<td>All Java development activities</td>
</tr>
</tbody>
</table>

Description

1. Number of Variable Per Line
   - One declaration of a variable per line is recommended since it encourages commenting.

2. Placement
   - Put declarations only at the beginning of blocks. (A block is any code surrounded by curly braces "{" and "}"). Don't wait to declare variables until their first use; it can confuse the unwary programmer and hamper code portability within the scope.
   - One exception to the rule is indexes of for loops, which in Java can be declared in the for statement syntax:

```
for (int i = 0; i < arraySize; i++)
{ ...
```

   - Do not make local variable declarations that hide variable declarations in enclosing blocks or variables in the declaring class or any base class.

3. Initialization
   - Try to initialize local variables where they're declared. The only reason not to initialize a variable where it's declared is if the initial value depends on some computation occurring first.
   - Avoid assigning several variables to the same value in a single statement. It is hard to read.
   - Do not use embedded assignments in an attempt to improve run-time performance. This is the job of the compiler, and besides, it rarely actually helps.

Rationale

Following the guidelines on declaration and initialization of variables in Java helps to improve readability and maintainability of source code.

Examples

Right

Code

1. Number of Variable Per Line
QAD Development Standards

```c
int level = 0;  // indentation level
int size = 0;   // size of table
```

Note - The examples above use one space between the type and the identifier. Another acceptable alternative is to use tabs, e.g.:

```c
int level;      // indentation level
int size;       // size of table
Object currentEntry;  // currently selected table entry
```

2. Placement

```c
void setItemProperty()
{
    int level = 0;  // beginning of method block
    if (condition)
    {
        int size = 0;  // beginning of "if" block
        ...
    }
}
```

3. Initialization

```c
int level = 0;  // level of item
int size = 0;   // size of table
```

Example of rewriting an embedded statement - see “Wrong Code” 3. Initialization for the corresponding incorrect assignment.

```c
totalAmount = totalDiscount + itemDiscount;
```

Wrong

Code

1. Number of Variable Per Line

```c
int level, size;
private Class myClass; private String myClassName;
```

2. Placement

Do not declare the same variable name in an inner block:

```c
int count;
...
private void addIndexProperty()
{
    if (condition)
    {
        int count;  // Not Allowed!
        ...
    }
    ...
}
```

3. Initialization

```c
fooBar.fChar = barFoo.lchar = 'c';  // AVOID!
```

Example of incorrect embedded assignment:

```c
totalAmount = (totalDiscount = customerDiscount + itemDiscount) + itemTax;
```

// AVOID!
See Also
STD-0249 Defining and Using Methods in Java

Summary

Follow the guidelines described in the Detailed Description section on defining and using methods in Java.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0249</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Java</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Java Programs, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All Java development activities</td>
</tr>
</tbody>
</table>

Description

1. Referring to Class Variables and Methods
   - Do not use an object to access a class (static) variable or method. Use a class name instead.
   - For accessing methods of the same class, prefix the method with ‘this’.

2. Returning values
   - Try to make the structure of your program match the intent. Example:

     ```java
     if (booleanExpression)
     {
         return true;
     }
     else
     {
         return false;
     }
     ```

     ...Instead of writing as

     ```java
     return booleanExpression;
     ```

     - If the return value involves complex computing, assign computed values to a variable and return the variable at the end.

Rationale

Following the guidelines on defining and using methods in Java helps to improve readability and maintainability of source code.

Examples

Right

Code

1. Referring to Class Variables and Methods
   Class Variable:
   Note: In the following example, INFO_Managers is a class variable of IEnvironmentManager
try {
  String mgrs = pEnvMgr.getInfo(IEnvironmentManager.INFO_Managers);
  if (mgrs != null) {
    managersFile = mgrs;
  }
} ...

Method from same class with 'this' prefix:

```java
public ApplicationManager(IEnvironmentManager pEnvMgr)
  throws FailureException, RemoteException {
  this.addManager(pEnvMgr);
  RootManager aManager = null;
  ...
}
```

2. Returning values

```java
boolean isSimpleItem(String pItem, boolean pSort, boolean pFilter) {
  boolean aReturnValue = true;
  ...
  if (pSort && !aSimpleDesc.isSortable()) {
    aReturnValue = false;
  } else {
    aReturnValue = true;
  }
  return aReturnValue;
}
```

See Also

STD-0249 Defining and Using Methods in Java
STD-0251 Code Blocks in Java

Summary

Follow the guidelines described in the Detailed Description section on using code blocks in Java.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0251</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Java</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Java Programs, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All Java development activities</td>
</tr>
</tbody>
</table>

Description

When coding Java code blocks, use the following formatting rules. The rule is applicable to class/interface declarations as well as method bodies, loops and other places a code block appears.

- Open brace "{" appears at the beginning of the next line at the same level of a declaration.
- The closing brace "}" starts a line by itself indented to match its corresponding opening brace.

Following the guidelines on using code blocks in Java helps to improve readability and maintainability of source code. Placing open and close braces on a separate lines makes the block easier to maintain.

Examples

Right

Code

```java
public class StatusMessage
{
    public StatusMessage(int pSeverity, String pItem, String pText, int pMsgNumber)
    {
        mySeverity = pSeverity;
        myItem = pItem;
        myText = pText;
        myMessageNumber = pMsgNumber;
    }

    ...  
}
```

Wrong

Code

```java
public class StatusMessage
{
    public StatusMessage(int pSeverity, String pItem, String pText, int pMsgNumber)
    {
        mySeverity = pSeverity;
        myItem = pItem;
        myText = pText;
        myMessageNumber = pMsgNumber;
    }

    ...  
}
```
public class StatusMessage {  
  //WRONG

  public StatusMessage(int pSeverity, String pItem, String pText, int pMsgNumber) {  
    //WRONG
    mySeverity = pSeverity;
    myItem = pItem;
    myText = pText;
    myMessageNumber = pMsgNumber;
  }

  ...

  ...

}

See Also
STD-0254 Localization Tags Do not place 2 tags consecutively

Summary

Localization Tags: Do not place 2 tags consecutively

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0254</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Localizations</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

When adding localization tags, do not place 2 tags right after each other.

Rationale

The intent is to implement as few localization tags as possible to not affect readability of source code too much.

Examples

In the wrong method example you see that 2 tags {&EXAMPLE-P-TAG2} and {&EXAMPLE-P-TAG3} are placed consecutively.

They have to be replaced by 1 tag {&EXAMPLE-P-TAG2} only. See the right method example.

The last tag is renamed from {&EXAMPLE-P-TAG4} into {&EXAMPLE-P-TAG3}

Right

Code

```/*N0FB*/   {&EXAMPLE-P-TAG1}
form
   check_nbr
   cust base_rpt
   cm_sort
   with frame a.

   /* Set Frame Labels */
   setFrameLabels(frame a:handle)

/*N0FB*/   {&EXAMPLE-P-TAG2}
   display check_nbr
   cm_addr @ cust base_rpt
   cm_sort
   with frame a.

/*N0FB*/   {&EXAMPLE-P-TAG3}
```

Content of cxcustom.i:
Content of localization include files:

```c
/* cx0059.i */
form check_nbr
    cust base_rpt
    l_bank
    l_pm
    cm_sort
    with frame a.

/* cx0060.i */
display check_nbr
    cm_addr @ cust base_rpt
    l_bank
    l_pm
    cm_sort
    with frame a.
```

Wrong Code

```c
/*N0FB*/ {&EXAMPLE-P-TAG1}
form
    check_nbr
    cust base_rpt
    cm_sort
    with frame a.

/*N0FB*/ {&EXAMPLE-P-TAG2}
/* Set Frame Labels */
setFrameLabels(frame a:handle)

/*N0FB*/ {&EXAMPLE-P-TAG3}
display
    check_nbr
    cm_addr @ cust base_rpt
    cm_sort
    with frame a.

/*N0FB*/ {&EXAMPLE-P-TAG4}
```

Content of cxcustom.i:

```c
&GLOBAL-DEFINE EXAMPLE-P-TAG1 &IF FALSE &THEN
&GLOBAL-DEFINE EXAMPLE-P-TAG2 &ENDIF ~(cx0059.i~)
&GLOBAL-DEFINE EXAMPLE-P-TAG3 &IF FALSE &THEN
&GLOBAL-DEFINE EXAMPLE-P-TAG4 &ENDIF ~(cx0060.i~)
```

Content of localization include files:
/* cx0059.i */
form check_nbr
  cust base_rpt
  l_bank
  l_pm
  cm_sort
with frame a.
/* cx0060.i */
display check_nbr
  cm_addr @ cust base_rpt
  l_bank
  l_pm
  cm_sort
with frame a.

See Also
STD-0256 Localization Tags List tags alphabetically in cxcustom.i

Summary

Localization Tags: List tags alphabetically in cxcustom.i

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Localizations</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

In cxcustom.i the localization tags (global-define's) need to be listed in alphabetical order.

Rationale

When the list of tags in cxcustom.i grows, it takes more time to find the tag wanted. Therefore the tags should be listed in alphabetical order.

Examples

Right

Code

&GLOBAL-DEFINE ARPAIQ-P-TAG1 ~{cx0032.i~}
&GLOBAL-DEFINE ARPAIQ-P-TAG2 ~{cx0033.i~}
&GLOBAL-DEFINE ARPAIQ-P-TAG3 &IF FALSE &THEN
&GLOBAL-DEFINE GLTRIQ1C-P-TAG1 ~{cx0031.i~}
&GLOBAL-DEFINE WOWORCA-P-TAG1 ~{xlt10026.i~}

Wrong

Code

&GLOBAL-DEFINE GLTRIQ1C-P-TAG1 ~{cx0031.i~}
&GLOBAL-DEFINE WOWORCA-P-TAG1 ~{xlt10026.i~}
&GLOBAL-DEFINE ARPAIQ-P-TAG1 ~{cx0032.i~}
&GLOBAL-DEFINE ARPAIQ-P-TAG2 ~{cx0033.i~}
&GLOBAL-DEFINE ARPAIQ-P-TAG3 &IF FALSE &THEN

See Also

STD-0240 Localization Tags
STD-0258 Localization Tags Do not use a tag include file when tag is only a few statements
STD-0259 Localization Tags Avoid placing tags in form statements
STD-0260 Localization Tags Avoid placing
STD-0261 Localization Tags Avoid placing tags in display statements
STD-0262 Localization Tags Standard for placing tags in find-for each-where statements
STD-0258 Localization Tags

Do not use a tag include file when tag is only a few statements

Summary

Do not use a tag include file, when the tag content is only a few statements of code.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0258</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Localizations</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Do not use a tag include file, when the tag content is only a few statements of code. In that case it is better for the readability to include the tag content directly in cxcustom.i.

Rationale

Keeping the source code better to read. The more nesting of code, the more complex it is to investigate which will take more work and introduce errors more quickly.

Examples

Right

Code

```c
&GLOBAL-DEFINE ARPAIQ-P-TAG2  l_bank l_pm
&GLOBAL-DEFINE APRVMT-P-TAG2 wait_exp_mt = no.
&GLOBAL-DEFINE ETCOUNT-P-TAG6 &ENDIF cnttot = 260
```

Wrong

Code

```c
&GLOBAL-DEFINE ARPAIQ-P-TAG2 ~(cx0033.i~)
&GLOBAL-DEFINE APRVMT-P-TAG2 ~(cx0044.i~)
&GLOBAL-DEFINE ETCOUNT-P-TAG6 &ENDIF ~(cx0006.i~)

/* Content of associated include files */
/* cx0006.i */
cnttot = 260

/* cx0033.i */
l_bank
l_pm

/* cx0044.i */
wait_exp_mt = no.
```
See Also

STD-0240 Localization Tags
STD-0256 Localization Tags List tags alphabetically in cxcustom.i
STD-0258 Localization Tags Do not use a tag include file when tag is only a few statements
STD-0259 Localization Tags Avoid placing tags in form statements
STD-0260 Localization Tags Avoid placing tags in include statements
STD-0261 Localization Tags Avoid placing tags in display statements
STD-0262 Localization Tags Standard for placing tags in find-for each-where statements
STD-0259 Localization Tags Avoid placing tags in form statements

Summary

Localizing Tags: Avoid placing tags in form statements

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0259</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Localizations</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

When a form needs to be changed using localization tags, an option might be to simply add the tag within the form statement at the specific place where this is required. Disadvantage of this approach is however that it can result in several tags within a single form statement, depending on the intended changes that need to be made, such as changing a label or a format, additional fields or changes in frame appearance.

Instead, place 1 tag before the form statement and another tag after the form statement. Copy the complete form statement to a customization include and change the form statements as needed. This method has another advantage. Both new tags can be used for other purposes in other localizations.

The disadvantage of this method is that any changes made in the standard form statement (for instance because of changes in the baseline) must be copied into the changed form statement.

Rationale

Less localization tags are needed.

Examples

Right

Code
Wrong

Code

```plaintext
form
   arautosel.sel_idr /*format "X(1)"*/ label {&arpamte_p_6}
   arautosel.ar_nbr format "X(8)"
   arautosel.ar_type
   arautosel.ar_due_date
   arautosel.balance label {&arpamte_p_4}
   arautosel.applied label {&arpamte_p_3}
   arautosel.discount label {&arpamte_p_1}
with frame c width 80
title color normal
   (getFrameTitle("AUTOMATIC_PAYMENT_SELECTION",38)).
```

See Also

N/A
STD-0260 Localization Tags Avoid placing tags in include statements

Summary

Localization Tags: Avoid placing tags in include statements.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0260</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Localizations</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Place 1 tag before the form statement and 1 tag after the include statement. Copy the complete include statement to a customization include and change the include statement as needed. Don't place tags within the original include statement.

Rationale

The placements of tags within an include is arbitrarily. This can result in several tags within a single include, depending on changes to be made (parameters).

This method has another advantage. Both tags can be used for other purposes (other statements before and/or after the standard include). The disadvantage of this method that any changes made in the standard include statement must be copied into the localizaed include statement.

Examples

Right

Code

Arpamte.p

```c
/*N0FB*/ {&ARPAMTE-P-TAG1}
{swselect.i
  rest of include statement
}
/*N0FB*/ {&ARPAMTE-P-TAG2}
```

Wrong

Code

```c
/* Content of cxcustom.i related to example */
&GLOBAL-DEFINE ARPAMTE-P-TAG1 &IF FALSE &THEN
&GLOBAL-DEFINE ARPAMTE-P-TAG2 &ENDIF ~{cx0030.i~}
```
QAD Development Standards

/*N0FB*/
{&ARPAMTE-P-TAG1}
rest of include statement
}

/* Content of cxcustom.i related to example */
&ampGLOBAL-DEFINE ARPAMTE-P-TAG1 ~{cx0030.i~}

See Also

STD-0240 Localization Tags
STD-0256 Localization Tags List tags alphabetically in cxcustom.i
STD-0258 Localization Tags Do not use a tag include file when tag is only a few statements
STD-0259 Localization Tags Avoid placing tags in form statements
STD-0261 Localization Tags Avoid placing tags in display statements
STD-0262 Localization Tags Standard for placing tags in find-for each-where statements
STD-0261 Localization Tags Avoid placing tags in display statements

Summary
 Localization Tags: Avoid placing tags in display statements

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0261</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Localizations</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
 Instead of placing a tag within a display statement, place 1 tag before the display statement and 1 tag after the display statement. Doing this gives the following possibilities:

A display statement (or any other statement) can be placed before the standard display statement.
A display statement (or any other statement) can be placed after the standard display statement.
The standard display statement can be excluded and be replaced by the customized display statement (see also comments on form statement and include statement.

Normally the fields to be displayed will be grouped in a single display statement. But there is no problem spreading it over several display statements if it makes localization more flexible.

Rationale
 The placements of tags within a display statement is arbitrarily. This can result in several tags within a single include, depending on changes to be made, which may result in source code with many localization tags reducing the readability of the code.

Examples

Right

**Code**

/*N0FB*/ {&ARPAIQ-P-TAG6A}
display check_nbr cm_addr @ cust base_rpt cm_sort
with frame a.
/*N0FB*/ {&ARPAIQ-P-TAG6B}
cxcustom.i:

&GLOBAL-DEFINE ARPAIQ-P-TAG6A display "" @ l_bank "" @ l_pm with frame a.

OR

&GLOBAL-DEFINE ARPAIQ-P-TAG6B display "" @ l_bank "" @ l_pm with frame a.

OR
Wrong

Code

Arpaiq.p

display check_nbr cm_addr @ cust base_rpt cm_sort
/*N0FB*/
{&ARPAIQ-P-TAG6}
with frame a.

cxcustom.i:

/*cx0035.i - localization file */
"" @ l_bank "" @ l_pm

Localization include file:

See Also

STD-0240 Localization Tags
STD-0256 Localization Tags List tags alphabetically in cxcustom.i
STD-0258 Localization Tags Do not use a tag include file when tag is only a few statements
STD-0259 Localization Tags Avoid placing tags in form statements
STD-0260 Localization Tags Avoid placing tags in include statements
STD-0262 Localization Tags Standard for placing tags in find-for each-where statements
STD-0262 Localization Tags Standard for placing tags in find-for each/where statements

Summary

Localization Tags: Standard for placing tags in find/for each/where statements

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0262</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Localizations</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

For making changes to find, for each or where statements, multiple methods of placing tags are possible. To prevent that all these various methods of placing tags will emerge in the source code, a standard for placing these tags is suggested. It is suggested to always add three tags at the following places:

- Before the where clause
- After the where clause but before the lock and index clause
- After the lock and index clause

This method has the following possibilities:

- The standard where, lock and index clause can be excluded and be replaced by the customized where, lock and index clause using the first and third tag.
- The standard where clause can be excluded and be replaced by the customized where clause using the first and second tag.
- The standard lock and index clause can be excluded and be replaced by the customized lock and index clause using the second and third tag.

Rationale

The placements of tags within a find/for each/where statement is arbitrarily. Sometimes the where clause has to be changed or the lock status or the index. This can result in setting multiple tags within a find/for each/where statement. A standard tagging method should be used.

Examples

Right

Code

First example Arpaiq.p

```java
find prev ar_mstr
/*N0FB*/   {&ARPAIQ-P-TAG10A}
    where ar_bill = cust
    and ((ar_curr = base_rpt)
    or (base_rpt = ""))
/*N0FB*/   {&ARPAIQ-P-TAG10B}
    no-lock use-index ar_bill.
/*N0FB*/   {&amp;ARPAIQ-P-TAG10C}
```

Tags 10A and 10C are undefined in this example, only 10B is used.
Second example arpaiq.p

```c
find prev ar_mstr
/*N0FB*/   {&ARPAIQ-P-TAG16A}
   where ar_curr = base_rpt
   or base_rpt = ""
/*N0FB*/   {&ARPAIQ-P-TAG16B}
no-lock use-index ar_date.
/*N0FB*/   {&ARPAIQ-P-TAG16C}
```

Tags 16C is undefined in this example.

First example: arpaiq.p

```c
find prev ar_mstr where ar_bill = cust
      and ((ar_curr = base_rpt)
       or (base_rpt = ""))
/*N0FB*/   {&ARPAIQ-P-TAG10}
no-lock use-index ar_bill.
```

```c
&GLOBAL-DEFINE ARPAIQ-P-TAG10   ~(cx0037.i~)
/* cx0037.i – localization
file   */
   and (ar_bank = l_bank or l_bank = "")
   and (ar_chr05 = l_pm or l_pm = "")
```

Second example: arpaiq.p

```c
/*N0FB*/   {&ARPAIQ-P-TAG16}
   find prev ar_mstr where ar_curr = base_rpt
   or base_rpt = ""
   no-lock use-index ar_date.
/*N0FB*/   {&ARPAIQ-P-TAG17}
```

```c
&GLOBAL-DEFINE ARPAIQ-P-TAG16   &IF FALSE &THEN
&GLOBAL-DEFINE ARPAIQ-P-TAG17   &ENDIF ~(cx0041.i~)
/* cx0041.i – localization
file   */
   (ar_curr = base_rpt or base_rpt = "")
   and (ar_bank = l_bank or l_bank = "")
   and (ar_chr05 = l_pm or l_pm = "")
```
STD-0240 Localization Tags
STD-0256 Localization Tags List tags alphabetically in cxcustom.i
STD-0258 Localization Tags Do not use a tag include file when tag is only a few statements
STD-0259 Localization Tags Avoid placing tags in form statements
STD-0260 Localization Tags Avoid placing tags in include statements
STD-0261 Localization Tags Avoid placing tags in display statements
STD-0263 How to use the 'pxrun.i' construct

Summary

The 'pxrun.i' construct provides a rich set of parameters to execute internal procedures with various options. This Standard describes 'pxrun.i' in detail along with each of its parameters and their proper use.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0263</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Java UI, Procedure Structure</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities utilizing the Project X infrastructure</td>
</tr>
</tbody>
</table>

Description

Within the Project X infrastructure, the 'pxrun.i' construct is utilized to execute internal procedures instead of the Progress 'run' statement or 'gprunp.i'. (See the referenced standards below on when/why to use pxrun.i)

pxrun.i was constructed to support XUI development. While there is no technical reason preventing its use outside of XUI, unless specific features supplied by pxrun.i are necessary, other alternatives should be employed.

The general format of a call to pxrun.i is:

```plaintext
{pxrun.i &PROC = '<internal procedure name>'
 &PROGRAM = '<procedure name>'
 &HANDLE = <variable of type 'handle'>;
 &PARAM = "(<parameter list>)"
 &NOAPPERROR = true
 &CATCHERROR = true
 $FIELD-LIST = <comma separated list of fields for the error messages>
 $MESSAGE-LIST=<comma separated list of error message numbers>;
 &PARENT = true}
```

Following are descriptions of the full set of parameters supported by pxrun.i and their usage:

**&PROC:**
Required parameter, which specifies which internal procedure is being run. This parameter will always be specified.

**&PROGRAM:**
If the referenced &PROC is defined in the current procedure, this parameter is not required. Otherwise, include this parameter and specify the external procedure where the internal procedure is defined.

**&HANDLE:**
For performance purposes, it is extremely important that whenever the 'pxrun.i' &PROGRAM parameter is used, that the &HANDLE parameter also be used. If the specified 'handle' is available and valid, an existing persistent procedure will be executed. Otherwise, the specified program will be loaded and a new 'handle' will be established.

A major exception involves cases where 'database switching' is employed in multi-DB related logic. A program is executed in the context of the current database. When the context is 'switched' to a different database, it is necessary to execute this program with a new 'handle' to avoid errors. That is, a given 'handle' is only valid relative to a specific 'program' executed in the context of a specific 'database'.

**&PARAM:**
Use of this parameter is conditional. It depends on whether the called internal procedure contains any parameters. In essence, use of &PARAM must conform to the signature of the called procedure.

**&NOAPPERROR:**
This parameter is optional. It can be used to suppress the raising of the Progress error condition as a result of an application (logic) error in the called procedure. In this way, the user can check the return-value to determine
whether the call was successful and how to handle any errors. Any Progress errors which occur as part of the pxrun.i call will always raise the error condition. If used, the parameter must be written as &NOAPPERROR = true.

&catcherror:
This parameter is optional. Typically, the call to pxrun.i is surrounded with a "do on error undo, return error return-value" block, so that if the error condition is raised, the error will be passed up the chain of calling programs. If a calling program needs to "catch" the error condition, then set the &catcherror parameter to true. This will suppress the "do on error" block around the pxrun.i call and leave the calling program the option of handling the error condition. If used, the parameter must be written as &catcherror = true.

&field-list:
This parameter is optional. It should have a comma delimited list of field names. This parameter can be used when the error/warning message(s) are to be associated to a physical field. If this parameter is used, the field information will be passed to the client in XUI application.

&message-list:
This parameter is optional. It should have a comma delimited list of message numbers that can be generated from the calling procedure, which may need to be associated with a physical field. If this parameter is used, the &field-list is also to be provided.

&pARENT:
Use of this parameter causes the 'SUPER' procedure version of the referenced &PROC to be called. This facility was implemented to support customization of QAD supplied code. As of this Guideline, it has not been utilized within QAD code, and any such future use should be subject to Architectural review.

Rationale
This Guideline has been developed to assist developers understand the various components of pxrun.i and to facilitate effective use.

Example

Right

Code

&PROC
The parameter 'internalProcedure1' is defined as an internal procedure in 'abc.p'.

```plaintext
{pxrun.i &PROC='internalProcedure1' &PROGRAM='abc.p'
 &HANDLE=ph_abc
...<additional parameters>
...}
```

&PROGRAM
Parameter 'internalProcedure1' is defined as an internal procedure in 'abc.p'.
The following code fragment is from 'xyz.p':

```plaintext
{pxrun.i &PROC='internalProcedure1' &PROGRAM='abc.p'
 &HANDLE=ph_abc
...<additional parameters>
...}
```

&HANDLE
/* Include file to define a specified variable 'as handle' */
{pxphdef.i abc}

{pxrun.i &PROC='internalProcedure1' &PROGRAM='abc.p'
  &HANDLE=ph_abc
  ...
  <additional parameters>
  ...
}

/* The &HANDLE parameter to 'pxrun.i' cannot be used between the
2 'gpalias' calls */
{gprun.i "" gpalias3.p "" (newDB, output errorCode)
}
{pxrun.i &PROC='internalProcedure1' &PROGRAM='abc.p'
  ...
  <additional parameters except &HANDLE>;
  ...
}
{gprun.i "" gpalias3.p "" (previousDB, output errorCode)
}

&PARAM

/* Internal Procedure definition for internalProcedure1 in abc.p */
Procedure internalProcedure1:
  define input parameter inputParameter1 as character no-undo.
  define output parameter outputParameter1 as character no-undo.
  ...
  <remainder of procedure definition>
END PROCEDURE.

/* Call to InternalProcedure1 from xyx.p */
{pxrun.i &PROC='internalProcedure1' &PROGRAM='abc.p'
  &HANDLE= ph_abc
  &PARAM="(input iparam, output oparam)"
  <additional parameters >
  ...
}

&NOAPPERROR:
The following example will display the error message when the 'validateCountryCode' procedure raises the invalid
country code error.

do on error undo, return error {&GENERAL-APP-EXCEPT}:
{pxrun.i &PROC='validateCountryCode' &PROGRAM='adctxr.p'
  &HANDLE=ph_adctxr
  &PARAM="(input pCountryCode)"
  &NOAPPERROR=true
  &CATCHERROR=true
}
if return-value <> {&SUCCESS-RESULT} then
do:
  /* MESSAGE #861 - COUNTRY CODE DOES NOT EXIST */
{pxmsg.i
  &MSGNUM=861
  &ERRORLEVEL={&APP-ERROR-RESULT}}
return error {&APP-ERROR-RESULT}.
end. /* IF RETURN-VALUE <> {&SUCCESS-RESULT} */
end. /* do on error undo, return error {&GENERAL-APP-EXCEPT}: */
&CATCHERROR:
In the following example, if an error occurs in the ‘readData...’ internal procedure, the logic will correctly leave the
‘do ..’ block and immediately execute the ‘assign hasError ...’ statement.

```plaintext
do on error undo, leave :
   {pxrun.i &PROC = "'readData' + pApp"
      &PARAM = "(input bufferHandle, output newRecord)"
      &CATCHERROR = true
   }
end.
assign hasError = error-status:error
returnValue = return-value.
{pxrun.i &PROC='processFields' &PROGRAM='pxtools.p'
   &HANDLE=ph_pxtools
   &PARAM="(this-procedure)"
}
```

&FIELD-LIST:
The following example will send the field information to the client in XUI when any application error occurs in
‘validateSupplier’.

```plaintext
{pxrun.i &PROC = 'validateSupplier' &PROGRAM = 'popoxr.p'
   &HANDLE=ph_popoxr
   &PARAM = "(input pSupplierId)"
   &FIELD-LIST=po_mstr.po_vend
}
```

&MESAGE-LIST:
The following example will send the field information to the client in XUI when an application error that results in
message number 2 occurs in ‘validateSupplier’. If an application error results in a message number other than 2, no field information will be sent
to the XUI client.

```plaintext
{pxrun.i &PROC = 'validateSupplier' &PROGRAM = 'popoxr.p'
   &HANDLE=ph_popoxr
   &PARAM = "(input pSupplierId)"
   &FIELD-LIST=po_mstr.po_vend
   &MESSAGE-LIST="2"
}
```

Wrong

Code

&PROC
The parameter ‘internalProcedure2’ is NOT an internal procedure in ‘abc.p’.

```plaintext
{pxrun.i &PROC='internalProcedure2' &PROGRAM='abc.p'
   &HANDLE=ph_abc
   ...
   &lt;additional parameters&gt;
   ...
}
```

&PROGRAM
Parameter ‘internalProcedure1’ is defined as an internal procedure in ‘abc.p’. The following code fragment is from
‘abc.p’, so &PROGRAM is not required:
&HANDLE

/* No &HANDLE provided, and the following pxrun.i is not in abc.p */
{pxrun.i &PROC='internalProcedure1' &PROGRAM='abc.p'
  ...  
  <additional parameters - not including &HANDLE>
  ...  
}

/* The &HANDLE parameter to 'pxrun.i' cannot be used between the 
2 'gpalias' calls */
{gprun.i 
  "" 
  gpalias3.p 
  "" 
  (newDB, output errorCode)"
}

{pxrun.i &PROC='internalProcedure1' &PROGRAM='abc.p'
 &HANDLE=ph_abc
  ...  
  <additional parameters>
  ...  
}

{gprun.i ""gpalias3.p"" ""(previousDB, output errorCode)"
}

&PARAM

Mis-matched parameters

/* Internal Procedure definition for internalProcedure1 in abc.p */
Procedure internalProcedure1:
  define input parameter inputParameter1 as character no-undo.
  define output parameter outputParameter1 as character no-undo.
  ...
  <remainder of procedure definition>
  ...
END PROCEDURE.

/* Call to InternalProcedure1 from xyx.p */
{pxrun.i &PROC='internalProcedure1' &PROGRAM='abc.p'
 &HANDLE=ph_abc
 &PARAM="(input iparam1, 
 input iparam2, /* Parameter not
 defined */
 output oparam)" 
 <additional parameters >
 ...  
}

&NOAPERROR:
The following example will not display the error message when the 'validateCountryCode' procedure raises the invalid country code error.
do on error undo, return error {&GENERAL-APP-EXCEPT}:
{
pxrun.i &PROC="validateCountryCode" &PROGRAM="adctxr.p"
 &HANDLE=ph_adctxr
 &PARAM="(input pCountryCode)"
 /* Parameter &NOAPPERROR not used */
 &CATCHERROR=true
}
if return-value <> {&SUCCESS-RESULT} then do :
/* MESSAGE #861 - COUNTRY CODE DOES NOT EXIST */
pxmsg.i &MSGNUM=861
 &ERRORLEVEL={&APP-ERROR-RESULT}
return error {&APP-ERROR-RESULT}.
end. /* IF RETURN-VALUE <> {&SUCCESS-RESULT} */
end. /* do on error undo, return error {&GENERAL-APP-EXCEPT}: */

&CATCHERROR:

In the following example, when there is an error in the 'readData....'
internal procedure, the logic will not leave the 'do ..' block and immediately execute the 'assign hasError ...'
statement.

do on error undo, leave :
{
pxrun.i &PROC = "'readData' + pApp"
 &PARAM = "((input bufferHandle,
 output newRecord)"
 /* No &CATCHERROR Parameter */
}
end.
assign hasError = error-status:error
returnValue=return-value.
{pxrun.i &PROC='processFields' &PROGRAM='pxtools.p'
 &HANDLE=ph_pxtools
 &PARAM="(this-procedure)"
}

&FIELD-LIST:

The following example will not send the field information to the client in XUI when any application error occurs in
'validateSupplier'.
{
pxrun.i &PROC = 'validateSupplier' &PROGRAM = 'popoxr.p'
 &HANDLE=ph_popoxr
 &PARAM = "(input pSupplierId)"
 /* No &FIELD-LIST Parameter */
}

&MESSAGE-LIST:

The following example will NOT send the field information to the client in XUI when an application error that results
in message number 2 occurs in 'validateSupplier'.
{
pxrun.i &PROC = 'validateSupplier' &PROGRAM = 'popoxr.p'
 &HANDLE=ph_popoxr
 &PARAM = "(input pSupplierId)"
 /* Missing &FIELD-LIST - required when &MESSAGE-LIST used */
 &MESSAGE-LIST="2"
}

See Also

STD-0265 Use 'pxrun.i' in XUI code to execute internal procedures
STD-0264 How to use the 'pxfunct.i' construct

Summary

The ‘pxfunct.i’ construct provides several parameters to call user-defined functions with various options. This standard describes ‘pxfunct.i’ in detail along with each of its parameters and their proper use.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0264</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Java UI, Procedure Structure</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities utilizing the Project X infrastructure</td>
</tr>
</tbody>
</table>

Description

Within the Project X infrastructure, the ‘pxfunct.i’ construct is utilized to execute functions instead of direct calls with the Progress ‘dynamic-function()’ statement.

Following is the general format of ‘pxfunct.i’:

```plaintext
{pxfunct.i &FUNCTION = '<function name>'
 &PROGRAM = '<program name>'
 &HANDLE = <variable of type handle>;
 &PARAM = "<parameter list>"
 &PARENT = true}
```

&FUNCTION:
Required parameter specifying the name of the function being called.

&PROGRAM:
If the referenced &FUNCTION is defined in the current procedure, this parameter is not required. Otherwise, include this parameter and specify the procedure name where the function is defined.

&HANDLE:
For performance purposes, it is important that whenever the ‘pxfunct.i’ &PROGRAM parameter is used, that the &HANDLE parameter also be used. The specified ‘handle’, which should be provided by using ‘pxgetph.i’, must be valid and point to the appropriate program prior to executing pxfunct.i.

A major exception involves cases where ‘database switching’ is employed in multi-DB related logic. A program is executed in the context of the current database. When the context is ‘switched’ to a different database, it is necessary to execute this program with a new ‘handle’ to avoid errors. That is, a given ‘handle’ is only valid relative to a specific ‘program’ executed in the context of a specific ‘database’.

In summary, if not within the context of a DB switch, definitely DO USE &HANDLE.

If within the context of a DB switch, definitely DO NOT USE &HANDLE.

&PARAM:
Use of this parameter is conditional. It depends on whether the called function definition contains any parameters.

In essence, use of &PARAM must conform to the signature of the called function.

&PARENT:
Causes the ‘SUPER’ version of the function to be invoked.

Super procedure(s) which contain definitions for a particular function must exist if the parameter is used.

This parameter is included for customization support only, and has not been utilized in any QAD developed code.

The &PROGRAM and &HANDLE parameters are not allowed when &PARENT is used.
Rationale

This Guideline has been developed to assist developers in understanding the various components of pxfunct.i and to facilitate effective use.

Standard STD-0266 see References addresses the required use of pxfunct.i. This Guideline discusses How to use it properly.

Right

Code

&FUNCTION

The function `myFunction1` is defined in the current procedure.

```
FUNCTION myFunction1 RETURNS <return type>
  <function definition>
END FUNCTION.
...
{pxfunct.i &FUNCTION='myFunction1'
  ...
  <additional parameters>
  ...
}
```

&PROGRAM

The function `myFunction2` is defined in abc.p.

```
/* Include file to get a 'handle' for the specified program */
{pxgetph.i abc.p}

{pxfunct.i &FUNCTION='myFunction2' &PROGRAM = 'abc.p'
  &HANDLE = ph_abc
  ...
  &<additional parameters>&gt;
  ...
}
```

&HANDLE

The function `myFunction2` is defined in abc.p.

```
/* Include file to get a 'handle' for a program containing a referenced function */
{pxgetph.i abc.p}

{pxfunct.i &FUNCTION='myFunction2' &PROGRAM = 'abc.p'
  &HANDLE = ph_abc
  ...
  <additional parameters>
  ...
}
```

&PARAM
FUNCTION myFunction3 RETURNS logical
  (input pFunctParam as character):
  ...
  <remaining function definition>
  ...
  END FUNCTION.

/*Function called from xyz.p*/
define variable myParamValue as character.
  {pxphdef.i abc}
  ...
  {pxgetph.i abc.p}
  {pxfunct.i &FUNCTION='myFunction3' &PROGRAM='abc.p'
    &HANDLE=ph_abc
    &PARAM="input myParamValue"}

&PARENT

The version of the function 'myFunction4' as defined in 'abcSuper.p' will be invoked in the following example:

[Super Procedure 'abcSuper.p' exists for 'abc.p' and contains a definition for 'myFunction4']

  /* Include file to get a 'handle' for the specified program */
  {pxgetph.i abc.p}

  {pxfunct.i &FUNCTION='myFunction4'
    &PARENT = true}

Wrong

Code

&FUNCTION

There is no function definition for 'myFunction1' in the current procedure.

... 
  {pxfunct.i &FUNCTION='myFunction1'
  ...
  <additional parameters>
  ...
  }

&PROGRAM

The function 'myFunction2' is defined in abc.p, but no &PROGRAM parameter is used.

  /* Include file to get a 'handle' for the specified program */
  {pxgetph.i abc.p}

  {pxfunct.i &FUNCTION='myFunction2' &PROGRAM parameter>
   &HANDLE = ph_abc
  ...
  <additional parameters>
  ...
  }

&HANDLE

DO NOT use '&HANDLE' within the context of database switching.
old_db = global_db.
if old_db &lt;&gt; global_db then
(gprun.i "gpalias3.p" '(old_db, output err-flag)"
)
...

if si_db &lt;&gt; global_db then
(gprun.i "gpalias3.p" '(si_db, output err-flag)"
)
...

if old_db &lt;&gt; global_db then
(gprun.i "gpalias3.p" '(old_db, output err-flag)"
)

&PARAM
Mis-matched parameters in the function call.

/*Function definition in abc.p*/
FUNCTION myFunction3 RETURNS logical
(input pFunctParam1 as character
input pFunctParam2 as integer):
...
<remaining function definition>
...
END FUNCTION.

/*Function called from xyz.p*/
define variable myParamValue as character.
(pxphdef.i abc)
...
(pxgetph.i abc.p)
(pxfunct.i &FUNCTION='myFunction3' &PROGRAM='abc.p'
&HANDLE=ph_abc
&PARAM="input myParamValue")

&PARENT
A super-procedure version of the function 'myFunction4' will be NOT invoked in the following example.
Either
1) No super-procedure(s) for 'abc.' exist, or
2) Function 'myFunction4' is not defined in the super procedure(s)

See Also
STD-0266 Use 'pxfunct.i' to call functions in XUI code
STD-0268 Use 'pxphdef.i' and 'pxgetph.i' to improve XUI performance
STD-0265 Use 'pxrun.i' in XUI code to execute internal procedures

Summary

Use 'pxrun.i' instead of the Progress ‘RUN’ statement or ‘gprunp.i’ to execute internal procedures within ‘Project X’ programs.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0265</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Java UI</td>
</tr>
<tr>
<td>Applicability</td>
<td>Progress code for Java UI enabled modules</td>
</tr>
</tbody>
</table>

Description

The 'pxrun.i' construct performs functions required by XUI beyond the basic ‘RUN’ statement or ‘gprunp.i’.

As with 'gprunp.i', 'pxrun.i' supports a "HANDLE" parameter which allows an instance of the procedure already loaded persistently to be executed, providing significant performance advantages. Unique to 'pxrun.i' however is support for locating and executing Customized versions of standard QAD code. These Custom versions of code are referenced via the customization table "cz_mstr", and maintained by program "czmsmtmp.p". See the Project X Training Material for more detailed information. If the 'pxrun.i' construct is not used, 'custom' versions of code are not enabled.

'pxrun.i' should be used in both of the following contexts:

1) run internal procedures in persistently running programs
2) run internal procedures in the current program

Support for 'customization' is a requirement for XUI enabled applications, and this functionality is supported only through the use of 'pxrun.i'. Performance dictates that XUI applications properly utilize persistent procedures. While this can be accomplished with either gprunp.i or 'pxrun.i', there are additional parameters unique to 'pxrun.i' that are utilized by XUI applications.

Example

Right

Code

```{pxphdef.i adctxr}

PROCEDURE testProc:

{pxrun.i &PROC= 'validateCountryCode' &PROGRAM='adctxr.p'
 &HANDLE=ph_adctxr}

{pxrun.i &PROC= 'init'}

{pxrun.i &PROC= 'writeData'
 &PARAM="( input currLogBuffer)"}

END PROCEDURE.
```
Wrong

Code

```plaintext
PROCEDURE testProc:
  run init.
  run writeData(input currLogBuffer).
END PROCEDURE.
```

See Also

STD-0266 Use 'pxfunct.i' to call functions in XUI code
STD-0263 How to use the 'pxrun.i' construct
STD-0266 Use 'pxfunct.i' to call functions in XUI code

Summary

Required use of 'pxfunct.i' in XUI code.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Java UI</td>
</tr>
<tr>
<td>Applicability</td>
<td>Progress code for Java UI enabled modules</td>
</tr>
</tbody>
</table>

Description

The 'pxfunct.i' construct provides important added value beyond simply executing a referenced 'user-defined function'. It provides support for 'customized versions' of functions, just as 'pxrun.i' supports customized versions of programs. If the 'pxfunct.i' construct is not used, any 'custom' versions of functions are not enabled.

When 'pxfunct.i' is called, it generates a 'dynamic-function' statement based on passed parameters. If found, a custom version of the function is executed, else the standard version is used.

XUI enabled applications require support for customized 'functions'. This support is implemented by using 'pxfunct.i' to call 'functions' instead of directly utilizing the Progress 'dynamic-function' construct.

Use 'pxrun.i' to call dynamic-functions:

```plaintext
usingGRS = ((pxfunct.i &FUNCTION='isGRSInUse'
&PROGRAM='rqgrsxr.p'
&HANDLE=ph_rqgrsxr)).
```

Do not use 'dynamic-function' directly:

```plaintext
usingGRS = dynamic-function("isGRSInUse"
 in getHandle("rqgrsxr.p",false)).
```

See Also

STD-0266 Use 'pxfunct.i' to call functions in XUI code
STD-0263 How to use the 'pxrun.i' construct
STD-0267 Use 'pxmaster.i' in all XUI Space Controllers

Summary
Use 'pxmaster.i' in all XUI 'Space Controller' programs

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0267</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Java UI,Procedure Structure</td>
</tr>
<tr>
<td>Applicability</td>
<td>XUI Application Development</td>
</tr>
</tbody>
</table>

Description
A new include file (pxmaster.i) and related program (pxmaster.p) supersedes prior use of include files 'pxdtctrl.i' and 'pxdsctrl.i' in XUI Space Controllers. The new include file also includes these files, and performs additional functions to significantly improve XUI performance. In addition, the use of 'pxmaster.p' has alleviated segmentation problems commonly encountered with 'large' XUI Space Controllers.

Rationale
Previous implementation of XUI Space Controllers frequently encountered 'segmentation' problems as functionality was added to 'large' Space Controllers. The new implementation (with pxmaster.i,p) includes functionality to support performance improvements as well as alleviating the segmentation problem.

Example
Right

Code
/* ppptxu.p - Item Maintenance Data Space Controller */
  .
  .
  .
  {mfdeclre.i}

/* Replaced the pxdtctrl.i and pxdsctrl.i with pxmaster.i. */
{pxmaster.i}

{ppitxt.i}
{ppicxt.i}
{ppicxt1.i}
{icsixt.i}
{gpngxt.i}

Wrong

Code
See Also

N/A
STD-0268 Use 'pxphdef.i' and 'pxgetph.i' to improve XUI performance

Summary

Always use 'pxphdef.i' and 'pxgetph.i' in XUI Progress code to improve performance of XUI code.

ID
STD-0268

Version
0

Language
Progress

Status
Published

Categories
Java UI, Procedure Structure

Applicability
All XUI (JavaUI) enabled Progress development activities

Description

The Progress XUI infrastructure attempts to minimize performance degradation (vis-à-vis CHUI) by running procedures persistently, storing handles to the procedures on a Program stack, then using those handles when subsequent calls are made to the same program. The 'pxphdef.i' and 'pxgetph.i' include files are XUI components used to facilitate this process. While 'pxphdef.i' is relevant to both 'pxrun.i' and 'pxfunct.i', the 'pxgetph.i' construct is only relevant to 'pxfunct.i' calls that use a handle. 'pxrun.i' includes internal functionality to check for a valid handle and establish one if necessary.

Since these constructs support the effective use of handles, they are not relevant in situations where handles should not be used. This is the case within the context of DB 'switching', as noted in the linked standards in the References section. If a pxrun.i or pxfunct.i call does not use the &HANDLE parameter, then pxphdef.i and pxgetph.i are not relevant.

pxphdef.i:
The 'pxphdef.i' include file is used to declare variables 'AS HANDLE' to hold handles to those persistent procedures. For each 'pxrun.i' or 'pxfunct.i' statement which includes the &HANDLE parameter in an XUI program, a {pxphdef.i <program name>} must be included in the file header section, before any procedural code.

The only code which is in this include file is a 'define variable' statement:

```
define variable ph_{1} as handle no-undo.
```

pxgetph.i:
When a 'pxfunct.i' statement contains the &HANDLE parameter, then a {pxgetph.i <program name>} statement must precede the call to make the handle available. In addition, a 'pxphdef.i' statement must exist in the file header to declare the HANDLE variable. The 'pxgetph.i' statement must precede the relevant 'pxfunct.i' statement so that a handle is available when it is referenced. When the same program is used multiple times this can be accomplished either by placing a single 'pxgetph.i' at the top of the code, or by using multiple 'pxgetph.i' statements - placing each one immediately before a relevant 'pxfunct.i' statement.

'pxgetph.i' determines whether a referenced handle (which has been declared via pxphdef.i) is valid. If it is not, it then obtains a handle to the procedure:

```
if not valid-handle(ph_{1}) then
    ph_{1} = getProcHandle('{1}.p',false).
```

Unless these 2 constructs are properly used, along with the other related Project X infrastructure elements, performance will be impaired.

Rationale

Minimizing the number of times when it is necessary to obtain a new procedure 'handle' is a major factor affecting performance of XUI applications. Proper use of these constructs is central to achieving that objective.
Examples

Right

Code

Example for a pxrun.i call:

```plaintext
/* All required 'handle' declarations should be located in the 
in the file header */
{pxphdef.i <program 1>}
...
  {pxphdef.i pxtools}
...
  {pxphdef.i <program 'n'>}

... ensuing program statements ...

{pxrun.i &PROC='processFields' &PROGRAM='pxtools.p'
  &HANDLE=ph_pxtools
  &PARAM="(this-procedure)"
}
```

Example for a pxfunct.i call:

```plaintext
{pxphdef.i <program 1>}
...
  {pxphdef.i pxtools}
...
  {pxphdef.i <program 'n'>}

... ensuing program statements ...

/* Get the handle (if necessary) just before the function call */
{pxgetph.i pxtools}
newRecord = {pxfunct.i &FUNCTION='isNewRecord'
            &PROGRAM='pxtools.p'
            &HANDLE=ph_pxtools
            &PARAM="input buffer po_mstrDirty:handle"}.
```

Example 2 for pxphdef.i and pxgetph.i - procedure handle used in function call:
...{pxphdef.i adcoxd}
...

PROCEDURE readDataSetImp:

{pxgetph.i adcoxd}

{pxrun.i &PROC='readDataSetImp'
 &PARAM="(input 'Address,contacts'

, input
'adcoxd.p',
input
'ad_mstr',
input
'Contact',
input-output serverCookie,
input
retrieveAscending,
input
retrieveNext,
input
getQueryString(conditionQuery,ph_adcoxd,bufferHandle),
input
'ad_addr ad_attn ad_attn2 ad_ext ad_ext2 ad_fax ad_fax2 ad_phone ad_phone2',
input
getOrderingString(orderingQuery,ph_adcoxd),
output
more,
input
bufferHandle) "}
END PROCEDURE.

Wrong

Code

Example for a pxrun.i call:

```plaintext
==> Missing pxphdef.i statement
{pxrun.i &PROC='processFields' &PROGRAM='pxtools.p'
 &HANDLE=ph_pxtools
 &PARAM="(this-procedure)
```

Example for a pxfunct.i call:

```plaintext
/* Check to see if this record is new */
==> Missing pxgetph.i statement
newRecord = {pxfunct.i &FUNCTION='isNewRecord'
 &PROGRAM='pxtools.p'
 &HANDLE=ph_pxtools
 &PARAM="input buffer po_mstrDirty:handle").
```

Example 2 for pxphdef.i and pxgetph.i:
PROCEDURE readDataSetImp:

{pxrun.i &PROC='readDataSetImp'
 &PARAM="(input 'Address,contacts'
   'adcoxd.p',
   'ad_mstr',
   'Contact',
   input-output serverCookie,
   retrieveAscending,
   retrieveNext,
   getQueryString(conditionQuery,getHandle("adcoxd.p",false),bufferHandle),
   'ad_addr ad_attn ad_attn2 ad_ext ad_ext2 ad_fax ad_fax2 ad_phone ad_phone2',
   getOrderingString(orderingQuery,getHandle("adcoxd.p",false)),
   more,
   input
   bufferHandle) "}
END PROCEDURE.

See Also

STD-0265 Use 'pxrun.i' in XUI code to execute internal procedures
STD-0266 Use 'pxfunct.i' to call functions in XUI code
STD-0273 Proper use of Progress 'valid-handle' function with procedures

Summary

Whenever the valid-handle function is used to check for a persistent procedure, the TYPE and either the UNIQUE-ID or FILE-NAME attributes must also be used to ensure that you have the correct handle to the persistent procedure you wish to run.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0273</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Data Handling Logic</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

It is not sufficient to check just the valid-handle of the persistent procedure before executing a "run <internal-procedure> in <handle>" statement. The reason for this is that Progress reserves the right to reuse handles within a Progress session. In other words, the handle may be valid but may be pointing to a different persistent procedure. Progress introduced several attributes for procedure handles (UNIQUE-ID and FILE-NAME) which help to ensure that you have the correct handle to the persistent procedure you wish to use.

Software engineers, developers, and programmers should always include a check of the TYPE attribute of a handle for the value "PROCEDURE" whenever they use the valid-handle function to access a persistent procedure. In addition, the developer should also verify that the handle still points to the persistent procedure that is intended to be run. A check on the UNIQUE-ID or FILE-NAME attribute should validate the correct procedure.

The check on TYPE must occur before the check on UNIQUE-ID or FILE-NAME, because the latter attributes are not queryable on all types of handles. Progress may reuse a procedure handle for something else, such as a widget handle, which would cause a run-time error when an attempt is made to read UNIQUE-ID or FILE-NAME.

Using the UNIQUE-ID insures that the handle still points to the specific instance of the persistent procedure that was originally executed. FILE-NAME only insures that the procedure handle points to an instance of the same program name. Generally, FILE-NAME is sufficient if the design does not call for multiple instances of a persistent procedure. When checking FILE-NAME, the developer must supply the program and path that was used to first instantiate the persistent procedure. For example, if gprun.i was used to instantiate the persistent procedure, subsequent checks on FILE-NAME should prefix the program name with the user's language directory and the program's 2-letter module directory. Using UNIQUE-ID, however, does require that this attribute is saved in a variable when the persistent procedure is first instantiated.

REFERENCE:
Please see Progress KBase entries #15673, #16355, and #20363 for further information. Please note that although these entries discuss primarily Windows environments, the problem can occur on any platform.

You may also read the discussion thread entitled “VALID-HANDLE and persistent procedures” at the web site www.progresstalk.com.

Rationale

This standard will ensure that you have the correct handle to the persistent procedure you wish to run.

Examples

Right
Code

Example 1 (using unique-id):

```lisp
&scoped-define pp_db_prefix DB=
&scoped-define pp_delimiter ;

@if defined(runp_pp_tag_{1}_{2}) = 0 &then
  &global-define runp_pp_tag_{1}_{2}
  define new global shared variable runp_h_{1}_{2} as handle no-undo.
/* Define an integer variable for the unique ID */
  define new global shared variable unid_h_{1}_{2} as integer no-undo.
&endif

if valid-handle(runp_h_{1}_{2}) and
  (runp_h_{1}_{2}:private-data matches
   "*{&pp_db_prefix}" + global_db + "{%pp_delimiter}""
  <> true
then do:
  /* Delete the persistent procedure */
  if valid-handle(runp_h_{1}_{2}) then delete object runp_h_{1}_{2}
  no-error.
  assign
  /* When deleting a persistent procedure, initialize the */
  /* Unique ID to 0 */
  unid_h_{1}_{2} = 0
  runp_h_{1}_{2} = ?.
end. /* if */

if valid-handle(runp_h_{1}_{2}) and
  (runp_h_{1}_{2}:private-data matches
   "*{&pp_db_prefix}" + global_db + "{%pp_delimiter}""
  <> true
then do:
end. /* if */

/* If persistent procedure is not instantiated, then do so. Keep */
/* track of the DB against which the procedure is instantiated. */
if not valid-handle(runp_h_{1}_{2})
/* Ensure that the handle is of type Procedure */
  or (runp_h_{1}_{2}:type <> "PROCEDURE")
/* Ensure that the unique-id matches saved value */
  or (runp_h_{1}_{2}:unique-id <> unid_h_{1}_{2})
then do:
  {gprun.i ""{1}.{2}"" "persistent set runp_h_{1}_{2}" }
  assign
  /* Assign a unique-id to the handle */
  unid_h_{1}_{2} = runp_h_{1}_{2}:unique-id
  runp_h_{1}_{2}:private-data =
    "*{&pp_db_prefix}" + global_db + "{%pp_delimiter}".
end.
/* Run the internal procedure within external persistent procedure */
run {3} in runp_h_{1}_{2} {4}.
```

Example 2 (using file-name):
define variable program-name as character no-undo.

program-name = global_user_lang_dir + substring("{1}",1,2) + '/' + "{1}.{2}".

/* If persistent procedure is not instantiated, then do so. Keep */
/* track of the DB against which the procedure is instantiated. */
if not valid-handle(runp_h_{1}_{2})
/* Ensure that the handle is of type Procedure */
or (runp_h_{1}_{2}:type &lt;&gt; "PROCEDURE")
/* Ensure that the file-name matches program */
or (runp_h_{1}_{2}:file-name <> program-name)
then do:
   (gprun.i "**{1}.{{2}}** "persistent set runp_h_{1}_{2}"")
   runp_h_{1}_{2}:private-data =
      "(&pp_db_prefix)" + global_db + "&pp_delimiter".
end.

Wrong

N/A

See Also

N/A
STD-0274 Do not use 'U' on literal strings

Summary
Do not use "U" on literal strings.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0274</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
Do not use "U" on literal strings.

The "U" statement was previously used for the Progress Translation Manager to indicate the literal string did not have to be translated. With the advent of the new Externalized Label project, the Progress Translation Manager is no longer used.

Examples

Right

Code

```c
if c-application-mode <> 'web' then return.
```

Wrong

Code

```c
if c-application-mode <> 'web':U then return.
```

See Also
N/A
STD-0275 Use getTermLabel() function variants to access literal strings in the code (eB and up)

Summary

Use getTermLabel() function or its variants to access literal strings from the Label Master.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0275</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Translatable Strings</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities (MFG/PRO versions eB and up)</td>
</tr>
</tbody>
</table>

Description

Literal strings in the code should be accessed by using getTermLabel() function or its variants depending on the usage of the string. Literal strings are the text strings not associated with a field or a variable.

List of getTermLabel() function variants:
1. getTermLabel Returns a label for a specified label term in a string that is no longer than the specified length.
2. getTermLabelCentered Returns a label for a specified label term, centered within the specified length (and padded with white space).
3. getTermLabelFillCentered Returns a label for a specified label term, centered within the specified length (the head and trail of which is filled with any specified character).
4. getTermLabelRt Returns a right-justified label for a specified label term in a string that is no longer than the specified length.
5. getTermLabelRtColon Returns a right-justified label with an appended colon for a specified term in a string that is no longer than the specified length.

In PUT statement, use gplblfmt.i to format the string based on the label retrieved by getTermLabel function or its variants. If the length available for the string is not fixed, it is necessary to have a dynamic calculation of the string’s length. This allows a string to be longer or shorter depending on its translation.

Rationale

Literal strings will not be translated into foreign language if getTermLabel() function or its variants are not used.

Example

Right

Code

Example 1:
define variable titl as character format "x(132)" no-undo.

form
bk_code colon 13
ad_date colon 65
bk_desc colon 13
ad_lang colon 65
ad_sort colon 13
with frame b side-labels width 132 title color normal titl.

titl = dynamic-function('getTermLabelFillCentered' in h-label,
input "BANK_ADDRESS",
input 132,
input "+")

Example 2: display format for field wo_part is "x(18)", so the maximum length passed to getTermLabel function is 18.

display getTermLabelRtColon("REPORT_TOTAL",18) @ wo_part
with frame b.

Example 3: Use gpblfmt.i to format strings at runtime depending on the length of translated string.

put {gpblfmt.i
  &FUNC=getTermLabel(""SITE"",8)
  &CONCAT="': '"
}
op_site " 
{gpblfmt.i
  &FUNC=getTermLabel(""WORK_CENTER"",14)
  &CONCAT="': '"
}
op_wkctr " 
{gpblfmt.i
  &FUNC=getTermLabel(""MACHINE"",10)
  &CONCAT="': '"
}
op_mch skip.

Wrong Code

Example 1:

/* ********** Begin Translatable Strings Definitions ********** */

&amp;SCOPED-DEFINE apbkrp_p_2 " Bank Address "
/* MaxLen:16 Comment: Frame Title for Bank Address */

/* ********** End Translatable Strings Definitions ********** */

define variable titl as character format "x(132)" no-undo.

form
bk_code colon 13
ad_date colon 65
bk_desc colon 13
ad_lang colon 65
ad_sort colon 13
with frame b side-labels width 132 title color normal titl.

titl = fill("-",58) + (&apbkrp_p_2) + fill("-",58).
Example 2:

```plaintext
/* ********** Begin Translatable Strings Definitions ********** */

&SCOPED-DEFINE woworp9a_p_12 "Report Total:
/* MaxLen:18 Comment: Print Report Total Amount. */

/* ********** End Translatable Strings Definitions ********** */

display {&woworp9a_p_12} @ wo_part
with frame b.
```

Example 3:

```plaintext
/* ********** Begin Translatable Strings Definitions ********** */

&SCOPED-DEFINE sfwcrp03_p_9 "Site: 
/* MaxLen: Comment: */

&SCOPED-DEFINE sfwcrp03_p_11 "Work Center: 
/* MaxLen: Comment: */

&SCOPED-DEFINE sfwcrp03_p_14 "Machine: 
/* MaxLen: Comment: */

/* ********** End Translatable Strings Definitions ********** */

put {&sfwcrp03_p_9} op_site " 
{&sfwcrp03_p_11} op_wkctr " 
{&sfwcrp03_p_14} op_mch skip.
```

See Also

N/A
STD-0276 Source file formatting - no commented out code

Summary

Do not leave commented out code in source files.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0276</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>All</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Source files should not contain inactive code that has been commented out. With the institution of a configuration management (CM) system, it is no longer necessary to preserve all old code in a source file. This was originally done so that a programmer could see the history of modifications made to a source file. A CM system gives the programmer the ability to see such a history by comparing versions of a source file and highlighting the differences. This is a preferable solution to the problem, since commented out code makes source files very difficult to read, and have led programmers to create programs specifically for stripping out such comments.

Additionally, the development tool pxtclean.p will automatically remove commented code from Progress source files. Nonetheless, it is preferable not to rely upon this tool for removing inactive code.

Examples

Right

Code

```
find sct_det exclusive-lock
where sct_sim = {&set} and sct_part = {&part}
and sct_site = lsc_site no-error.
```

Wrong

Code

```
find sct_det exclusive-lock
where sct_sim = {&set} and sct_part = {&part}
/* and sct_site = {&site} no-error. */
and sct_site = lsc_site no-error.
```

See Also

STD-0198 Source file formatting - Patch markers
STD-0277 Always use pxmsg.i when calling messages (eB and up)

Summary
Always use pxmsg.i to invoke a message in MFG/PRO. Do not use mfmsg*.i procedures in Release eB and above.

ID       STD-0277
Version   2
Language  Progress
Status    Published
Categories Character UI Apps, Procedure Structure
Applicability All development activities

Description
New code in MFG/PRO eB and above should always use pxmsg.i for displaying messages.

Prior to version eB of MFG/PRO there were numerous procedures for invoking a message in MFG/PRO. In eB, the mfmsg*.i procedures (mfmsg.i, mfmsg01.i, mfmsg02.i, mfmsg03.i, mfmsg04.i, mfmsg07.i, mfmsg07g.i, mfmsg08.i, mfmsg09.i and mfmsg10.i) were consolidated into a single procedure named pxmsg.i. pxmsg.i supports all the various options needed for invoking messages previously found in the mfmsg*.i procedures.

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).

The message number and error level should appear on the same line as the call to pxmsg.i to assist with information gathering when using grep.

Instructions for using pxmsg.i follow below.

```plaintext
{pxmsg.i
 &MSGNUM= - Optional. The message number for the message in the msg_mstr file. Either &MSGNUM or &MSGTEXT must be specified.
 &MSGTEXT= - Optional. The message text to be displayed. Either &MSGNUM or &MSGTEXT must be specified.
 &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.
 &MSGARG1= - Optional. An argument which 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.
 &MSGARG2= - Optional. See &MSGARG1.
 &MSGARG3= - Optional. See &MSGARG1.
 &MSGARG4= - Optional. See &MSGARG1.
```
&MSGARG5= - Optional. See &MSGARG1.

&MSGBUFFER= - Optional. The name of a variable where the message text should be placed. If this parameter is specified, the message will not be displayed.

&CONFIRM= - Optional. 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= - Optional. Indicates the confirmation field's type.

&BUTTONDATA= - Optional. If this is a confirmation message then this parameter must be defined to specify the button data associated for the confirmation dialog.

&CALLINGPGM= - Optional. 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= - Optional. Indicates the name of the field being processed which caused this message. If this parameter is not supplied, the default value "" will be used.

&PAUSEAFTER= - Optional. If TRUE is specified, a "pause." statement will be inserted after the call to display the message. If &PAUSEAFTER is not specified, no pause will be done.

&CONTEXT= - Optional. Sometimes needed in API mode on confirmation messages. The &CONTEXT string should be used to identify the current context (e.g. recno, loop iteration)

NOTE: msg_temp - Used in confirmation messages. This variable is defined in mfdeclre.i as local variable.

The API framework requires the use of 'pxmsg.i' to support an infrastructure for error message 'trapping' and 'logging'. All future ECommerce 'gateways' will also use the API framework, including pxmsg.i functionality.

- Code restructured for the XUI environment must use pxmsg.i.
- All mfmsg*.i procedures have been changed to call pxmsg.i in release eB and above.
- The new development tool pxtclean.p will automatically replace mfmsg.i in Progress source files for eB2 with pxmsg.i. Nonetheless, it is preferable not to rely upon pxtclean.p for this replacement.

Examples

Right

Design

This example illustrates the how to use pxmsg.i. It is the correct solution for both Example 1 and 2 listed below in
the Wrong section.

Code

```c
for first si_mstr fields(si_db si_site si_desc si_gl_set)
  where si_site = input slr_inventory_site no-lock: end.

  if not available si_mstr then do:
    /* SITE DOES NOT EXIST */
    (pxmsg.i &MSGNUM=708 &ERRORLEVEL={&APP-ERROR-RESULT})
    undo, retry.
  end.
```

Wrong

Code

Example 1 – code still uses mfmsg.i

```c
for first si_mstr fields(si_db si_site si_desc si_gl_set)
  where si_site = input slr_inventory_site no-lock: end.

  if not available si_mstr then do:
    (mfmsg.i 708 3) /* SITE DOES NOT EXIST */
    undo, retry.
  end.
```

Example 2 – code uses pxmsg.i but with an incorrect code format

```c
for first si_mstr fields(si_db si_site si_desc si_gl_set)
  where si_site = input slr_inventory_site no-lock: end.

  if not available si_mstr then do:
    /* SITE DOES NOT EXIST */
    (pxmsg.i
     &MSGNUM=708
     &ERRORLEVEL={&APP-ERROR-RESULT})
    undo, retry.
  end.
```

See Also

STD-0222 Different types of messaging logic (pre-eB)
STD-0279 Always define buffers for database table access within internal procedures

Summary

Use explicitly defined buffers for all tables that have database access within an internal procedure.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0279</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Table Access &amp; Storage, Procedure Structure</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Use explicitly defined buffers for all tables that have database access (read or create) within an internal procedure to ensure proper record scoping. If a buffer is not defined, then in the case of an internal procedure, the record will be scoped to the program. In the case of a persistent procedure, the record will still be scoped to the whole program and it will stay in scope until the persistent procedure is killed which could mean the rest of the session.

You can use compile list to help you find procedures that are missing explicitly defined buffers. For example, if you wanted to know if the program popoxr.p defined all of the necessary tables, go to the progress editor and run this code:

```
compile popoxr.p listing popoxr.lst
```

Now look at the file popoxr.lst. Go to the bottom of the listing and look at the section where the Blk. type is Procedure and the line number is 0. You should have no buffers scoped to this level. Here is a sample on what you should see:

```
...a/91/dev/popoxr.p 2617 Do No
...a/91/dev/popoxr.p 2618 Do No
...a/91/dev/popoxr.p 2635 Procedure No Procedure validatePOEditStatus
...a/91/dev/popoxr.p 2649 Do No
...a/91/dev/popoxr.p 2650 Do No
...a/91/dev/popoxr.p 2658 Do No
...a/91/dev/popoxr.p 2672 Procedure No Procedure validateExistingPO
...a/91/dev/popoxr.p 2685 Do No
...a/91/dev/popoxr.p 2687 Do No
...a/91/dev/popoxr.p 0 Procedure No
...91/dev/mfdeclre.i 134 Do No
...vgprun/mfdecgul.i 59 Do No
...vgprun/mfdecgul.i 62 Repeat No
...vgprun/mfdecgul.i 63 Do No
...vgprun/mfdecgul.i 69 Do No
...91/dev/pxpmmgr.i 58 Do No
```

If a procedure was missing an explicitly defined buffer (like the code example in the WRONG section), the listing might look like this:
Rationale

If a buffer is not defined in the internal procedure, then the record will be scoped to the whole program. If a record had been in exclusive lock in a transaction, it will be downgraded to a share lock when the transaction ends if it is still in scope. If it’s not a persistent procedure, the share lock will be released when the program ends. In the case of a persistent procedure the share lock will be held until then end of the session or when the persistent procedure has been killed.

Example

Right

Code

Example 1: In this case the PO control file needs to be read in the procedure.
PROCEDURE createPurchaseOrder:

/*============================================================================*/

PROCEDURE createPurchaseOrder:

Purpose: Create the Purchase Order Header record.
Exceptions: NONE
Conditions:
  Pre: None.
  Post: po_mstr(c)
Notes:
History:

define input parameter pPOId as character no-undo.
define parameter buffer po_mstr for po_mstr.
define buffer bf-poc_ctrl for poc_ctrl.
define variable globalUserId as character no-undo.
do on error undo, return error {&GENERAL-APP-EXCEPT}:
  /*Get the Control file for various default values*/
  {pxrun.i &PROC='readPOControl'
    &PARAM="(buffer bf-poc_ctrl)"}
do on error undo, return {&GENERAL-APP-EXCEPT}:
create po_mstr.
  assign
    globalUserId
    = {pxfunct.i &FUNCTION='getCharacterValue'
      &PROGRAM="pxgblmgr.p"
      &PARAM="input 'global_userid'"}
    po_nbr = pPOId
    po_ord_date = today
    po_due_date = today
    po_tax_date = ?
    po_ship = bf-poc_ctrl.poc_ship
    po_bill = poc_bill
    po_confirm = yes
    po_user_id = globalUserId
    po_fst_id = poc_fst_id /*(GST exempt id for company)*/
    po_pst_id = poc_pst_id
    po_ers_opt = if poc_ers_proc then poc_ers_opt else "1".
  if recid(po_mstr) = -1 then.
end.

Example 2: In this case the PO control file was read before calling the procedure. This is a good choice when the
PO Control file needs to be used by many procedures and would eliminate reading the same record again.
PROCEDURE createPurchaseOrder :

/*------------------------------------------------------------------------------
Purpose: Create the Purchase Order Header record.
Exceptions: NONE
Conditions: Pre: None.
Post: po_mstr(c)
Notes: History:
/*------------------------------------------------------------------------------

define input parameter pPOId as character no-undo.
define parameter buffer po_mstr for po_mstr.
define parameter buffer poc_ctrl for poc_ctrl.
define variable globalUserId as character no-undo.

do on error undo, return error {&GENERAL-APP-EXCEPT}:
  do on error undo, return {&GENERAL-APP-EXCEPT}:
    create po_mstr.
    assign
      globalUserId = {pxfunct.i &FUNCTION='getCharacterValue'
        &PROGRAM='pxgblmgr.p'
        &PARAM="input 'global_userid'"}
    po_nbr = pPOId
    po_ord_date = today
    po_due_date = today
    po_tax_date = ?
    po_ship = poc_ctrl.poc_ship
    po_bill = poc_bill
    po_confirm = yes
    po_user_id = globalUserId
    po_fst_id = poc_fst_id /*(GST exempt id for company)*/
    po_pst_id = poc_pst_id
    po_ers_opt = if poc_ers_proc then poc_ers_opt else "1".
    if recid(po_mstr) = -1 then.
    end.

Wrong

Code
PROCEDURE createPurchaseOrder:

/*============================================================================*/

Purpose: Create the Purchase Order Header record.

Exceptions: NONE

Conditions:

Pre: None.
Post: po_mstr(c)

Notes:

History:

define input parameter pPOId as character no-undo.
define parameter buffer po_mstr for po_mstr.

define variable globalUserId as character no-undo.

/* Here we are missing: */
/* define buffer poc_ctrl for poc_ctrl. */
/* poc_ctrl will be scoped to the persistent procedure. */

do on error undo, return error {&GENERAL-APP-EXCEPT}:

/*Get the Control file for various default values*/
{pxrun.i &PROC='readPOControl'
 &PARAM="(buffer poc_ctrl)"}

do on error undo, return {&GENERAL-APP-EXCEPT}:
create po_mstr.
assign
globalUserId = {pxfunct.i &FUNCTION='getCharacterValue'
 &PROGRAM="pxgblmgr.p"
 &PARAM="input 'global_userid'")
po_nbr = pPOId
po_ord_date = today
po_due_date = today
po_tax_date = ?
po_ship = poc_ctrl.poc_ship
po_bill = poc_bill
po_confirm = yes
po_user_id = globalUserId
po_fst_id = poc_fst_id /*(GST exempt id for company)*/
po_pst_id = poc_pst_id
po_ers_opt = if poc_ers_proc then poc_ers_opt else "1".

if recid(po_mstr) = -1 then.
end.

See Also

N/A
STD-0280 Do not use 'in this-procedure' on run statements

Summary

Do not use 'in this-procedure' on run statements.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0280</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Do not use 'in this-procedure' on run statements. When running an internal procedure where the run statement and procedure are contained in the same program, do not add 'in this-procedure' as the procedure handle.

Starting with the OBCM project, inclusion of the 'in this-procedure' statement was generated automatically. The intent was to put the internal procedure name and it's parent procedure's handle on the same line in the source code. Now inclusion of the 'in this-procedure' statement is no longer necessary.

In fact, the 'in this-procedure' handle statements will be removed by the 'strip and beautify' utility used to cleanup code prior to inclusion into the eB2 baseline.

The 'in this-procedure' handle is the handle Progress will default for the RUN statement, so the addition of this code is extraneous.

Examples

Right

Code

```
run applyPayments.
```

Wrong

Code

```
run applyPayments in this-procedure.
```

See Also

STD-0263 How to use the 'pxrun.i' construct
STD-0265 Use 'pxrun.i' in XUI code to execute internal procedures
STD-0286 Index Names

Summary

Index names cannot be longer than 28 characters minus the length of the table name. There are also special naming conventions for indexes starting with an OID field.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0286</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Indexes</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

All QAD Products:

Length Limit
Index names cannot be longer than 28 characters minus the number of characters in the table name to which they belong.

For example, if the table name is "so_mstr", then any index name created for so_mstr is limited to a maximum of:

28 - length("so_mstr") = 28 - 7 = 21 characters.

This is to accommodate name conversions that take place when converting an application to run on Oracle or SQL Server through the Progress DataServer.

MFG/PRO Indexes:

Index Name Format:
Index names should begin with the table prefix, followed by an underscore and then some unique name within the table. The name should be written in lower-case. For example, if the table name is "so_mstr", then any index name created for this table should begin with "so_", except for the main OID index.

Main OID Index:
An index consisting of only the main OID field in the table is referred to as the main OID index and should be named exactly like the main OID field. For example, if the table name is "so_mstr", then the main OID field will be oid_so_mstr and the unique index in so_mstr that consists of only this OID field should be named "oid_so_mstr".

Foreign Key OID Index:
An index whose first component is a foreign key OID field is referred to as a foreign key OID index and the name should begin with the table prefix, followed by the foreign key OID field name. If the resulting index name exceeds the maximum length as calculated above, it should be truncated. Also see STD-0325: Use OID fields as foreign keys to relate qaddb tables (eB3 and up).

Financials Indexes:

Primary Index:
Every table requires a primary index named "Prim" containing only the "<table-name>_ID" field. For example:

```
ADD INDEX "Prim" ON "Profile"
AREA "FIN_IDX"
UNIQUE
PRIMARY
INDEX-FIELD "Profile_ID" ASCENDING
```

Other Indexes
Provide a non-primary but unique index on all tables when possible.

**Foreign Key Indexes:**

1. If parent-table \(<A>\) has a relation to child-table \(<B>\), then \(<B>\) will hold a field named "\(<A>_ID\)" that represents this relation. For example: Parent table \(\text{Posting}\) is related to child-table \(\text{PostingLine}\), so \(\text{PostingLine}\) contains field \(\text{Posting_ID}\) as a link back to the \(\text{Posting}\) table. Create an index containing this foreign-key field on table \(<B>\). For example:

   ```sql
   ADD INDEX "Posting" ON "PostingLine"
   AREA "FIN_IDX"
   INDEX-FIELD "Posting_ID" ASCENDING
   ```

2. If there are multiple such relations between these tables, then \(<B>\) will hold fields named "\(<\text{Functional-differentiation-Identification}>\text{<A>}_ID\)" that reflect the different relations. For example, child-table \(\text{CDocument}\) contains field \(\text{CreationUsr_ID}\) to link to parent \(\text{Usr}\) table's \(\text{Usr_ID}\) field. Create an index containing this foreign-key field on table \(<B>\). For example:

   ```sql
   ADD INDEX "CreationUsr" ON "CDocument"
   AREA "FIN_IDX"
   INDEX-FIELD "CreationUsr_ID" ASCENDING
   ```

**Rationale**

Oracle 10g and 11g allow index names up to 30 characters long. However, the real limit on index name size comes from the Progress Oracle DataServer conversion utility which takes each index defined in Progress and prefixes it with \(<\text{tablename}>##\) to form the Oracle index name. This results in an Oracle index name following the convention of \(<\text{tablename}>##<\text{indexname}>\). The 30 character Oracle object name limit - ## = 28 characters. 28 - \(<\text{tablename}>\) leaves the maximum number of characters for the Progress index name. Table names less than 14 characters in length (see STD-0013: Table Name and Description) can have index names exceeding 14 characters as long as the combined \(<\text{tablename}>##<\text{indexname}>\) string does not exceed 30 characters.

**MFG/PRO Indexes:**

The main unique OID index should be readily identifiable by its name matching the table's main OID field (see DST-0005). Only one index in each table will start with "oid_".

Index names must be unique within the database (not just within the table, as Progress allows).

**All Other QAD Products, including Financials:**

Index names must be unique within the table, as Progress allows, but unique names within the database are strongly encouraged. This enables easier porting of an application to Oracle or SQL Server.

**Examples**

Assume that the so_mstr table now contains foreign key OID fields for the sold-to customer and the bill-to customer, \(\text{oid_cm_mstr}\) and \(\text{oid_cm_mstr_bill}\) respectively.

**Right**

Design
Wrong Design

<table>
<thead>
<tr>
<th>Add Index</th>
<th>ON</th>
<th>Area</th>
<th>Unique</th>
<th>Index-Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;oid_so_mstr&quot;</td>
<td>&quot;so_mstr&quot;</td>
<td>&quot;Schema Area&quot;</td>
<td>&quot;oid_so_mstr&quot;</td>
<td>ASCENDING</td>
</tr>
</tbody>
</table>

ADD INDEX "so_nbr" ON "so_mstr" (other index)
AREA "Schema Area"
UNIQUE
INDEX-FIELD "sod_domain" ASCENDING
INDEX-FIELD "sod_nbr" ASCENDING

ADD INDEX "so_oid_cm_mstr" ON "so_mstr" (foreign key OID index)
AREA "Schema Area"
INDEX-FIELD "oid_cm_mstr" ASCENDING (foreign key OID field)

ADD INDEX "so_oid_cm_mstr_bill" ON "so_mstr" (foreign key OID index)
AREA "Schema Area"
INDEX-FIELD "oid_cm_mstr_bill" ASCENDING (foreign key OID field)
INDEX-FIELD "so_nbr" ASCENDING
INDEX-FIELD "so_ord_date" ASCENDING

See Also

DST-0005 Developing with OID fields (eB3 and up)
STD-0013 Table Name and Description
STD-0014 Field Names
STD-0325 Use OID fields as foreign keys to relate qaddb tables (eB3 and up)
STD-0327 Every table must have a unique index & a primary index (eB3 and up)
STD-0288 Use lookups created via 'Browse Maintenance' - do not use scrolling window lookups

Summary
Use lookups created via "Browse Maintenance" - do not use scrolling window lookups.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0288</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps,Desktop UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>MFG/PRO eB and up</td>
</tr>
</tbody>
</table>

Description
This standard refers to the lookups/scrolling windows that are activated by using the HELP key in the UI when focus is on a particular field. A scrolling window is the old style lookup pre "Browse Maintenance". Scrolling windows should no longer be attached to a field as the HELP key lookup. Either reuse an existing lookup that was created via Browse Maintenance or create a new one with Browse Maintenance. To be able to take advantage of all the functionality of Browse maintenance review the guide in the "References" section of PLOT.

NOTE: There are also additional programs/include files that can handle some complexities. See STD-0292 and STD-0298. There is infrastructure in place within lookups created via Browse Maintenance that the Desktop HTML screens need in order to execute a lookup. The HTML screens cannot execute a scrolling window as the HELP key lookup.

See Also
STD-0298 Use gpbrparm.i to pass parameters to lookups
STD-0292 Use gpbranch.p to handle branched lookups
STD-0290 Do not use the Progress Browse widget in maintenance program frames

Summary

Do not use the Progress Browse widget in maintenance program frames.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0290</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Desktop UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>MFG/PRO eB and up</td>
</tr>
</tbody>
</table>

Description

This standard is not referring to browse widgets in lookups or power browses. It only refers to cases where the browse widget is being used in a maintenance frame. This is not common in MFG/PRO but it has been done.

Rationale

Desktop 2 has a Progress engine that walks the widget tree of an MFG/PRO screen collecting information about the widgets so that it can render them in html. The engine does not handle browse widgets. So, if a browse widget is used in a frame in MFG/PRO it will not appear in the Desktop 2 html screen.

Examples

Right

Design

Use MFG/PRO scrolling window include files to provide similar functionality as the browse widget provides. (NOTE: a scrolling window should NOT be used as the HELP key lookup but it can be used embedded in a maintenance program as indicated here.)

Code

Some examples of the use of scrolling window include files are:

```plaintext
ppptmt03.p "Item Cost Maintenance" (scrolling window include file: csscrad1.i)
fatrmt.p "Fixed Asset Transfers" (scrolling window include file: swselect.i)
```

Wrong

Design

Program qqbr.p "Export/Import Document Query" uses two browse widgets to provide the user with a list of Q/Linq documents and document data.

See Also

STD-0311 Progress selection-list widget and the Desktop
STD-0291 Do not separate labels from fields

Summary

Do not separate labels from fields

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0291</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Desktop UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>MFG/PRO eB and up</td>
</tr>
</tbody>
</table>

Description

It is possible in Progress to turn off the label of a field and replace it by adding to the frame a character variable that contains the label. This should be avoided. This applies to both side labels and column labels. There is an acceptable workaround for column labels which is illustrated in the examples below.

The Desktop 2 HTML engine will not be able to properly align or format the text of a label that is not attached to a field. The RIGHT example shows an acceptable solution on how to separate column labels from fields. In the WRONG example one variable is used to span multiple column labels. (Note: the examples do not follow the proper standards of retrieving translated text from the label master/detail records. This is done for simplicity and ease of illustration).

Examples

Right

Code

```plaintext
define variable x as character no-undo.
define variable y as character no-undo.
define variable z as character no-undo.
define variable columnLabel1 as character no-undo.
define variable columnLabel2 as character no-undo.
define variable columnLabel3 as character no-undo.

assign
columnLabel1 = "Field1"
columnLabel2 = "Field2"
columnLabel3 = "Field3".

form
  columnLabel1 columnLabel2 columnLabel3 skip x y z
  with frame a no-labels.

display columnLabel1 columnLabel2 columnLabel3 with frame a.

update x y z with frame a.
```

Wrong

Code
define variable x as character no-undo.
define variable z as character no-undo.
define variable columnLabel as character format "x(24)" no-undo.

columnLabel = "Field1 Field2 Field3".

form
  columnLabel skip
    x y z
  with frame a no-labels.

display columnLabel with frame a.
update x y z with frame a.

See Also

N/A
STD-0292 Use gpbranch.p to handle branched lookups

Summary

For fields that need to run one of two or more lookups based on some conditional logic use gpbranch.p.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0292</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Desktop UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>MFG/PRO eB and up</td>
</tr>
</tbody>
</table>

Description

"Window Help Maintenance" is referenced in this standard as the program to connect a lookup to a field. Keep in mind that for MFG/PRO eB2 and up the program is called "Drill Down/Lookup Maint".

In MFG/PRO there are certain fields that need to run different lookups depending on certain conditions. For example - a field may need one lookup under condition A and another lookup under condition B. One lookup will be hooked up to the field via "Window Help Maintenance" - this will be the lookup for condition A. gpbranch.p will then be used in the maintenance code to assign the appropriate lookup based on the condition. gpbranch.p sets a qad_wkfl record with the lookup information which is then used by the lookup launch mechanism to determine if the default lookup (attached via "Window Help Maintenance") should be used or whether an alternate lookup should be run.

There are two input parameters to gpbranch.p:

- **pFirstBrowse** - The name of the default lookup for the field that is specified as the "Procedure To Execute" in "Window Help Maintenance". This input parameter is mandatory and must always be supplied.
- **pSecondBrowse** - The name of the alternate lookup to run for the field. If this is blank then the default (pFirstBrowse) will be run.

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

Rationale

Previously this situation was handled by creating a wrapper lookup program that conditionally branched to one of the two lookups based on some data. This wrapper program was not a standard lookup created via Browse Maintenance. Because of this the Desktop 2 html screens were not able to run these lookups.

Example

In program povedrp.p there is a field (vendor) that needs lookup gplu360.p if variable, source_entry_code is set to "1" and needs lookup gplu348.p if variable, source_entry_code is any other value. There is a "Window Help Maintenance" record for this field with "Procedure To Execute" set to gplu360.p. The following RIGHT code example shows how gpbranch.p would be used in this situation.

Right

Code
if source_entry_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.

Wrong

Design

The wrong way to handle this would be to create a non standard lookup program (one not created via "Browse Maintenance") that had logic in it to conditionally launch the correct lookup.

See Also

STD-0298 Use gpbrparam.i to pass parameters to lookups
STD-0288 Use lookups created via 'Browse Maintenance' - do not use scrolling window lookups
STD-0293 Do not enable fields in more than one frame at a time

Summary

Limit enabled fields to only one frame at a time.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0293</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Desktop UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

The term “fields” in this standard is used to denote all widgets not just database fields - i.e. button widgets, combo-boxes, etc.

Generally in MFG/PRO fields are enabled via the set, prompt-for, or update statements. These statements allow fields to be enabled in only one frame at a time. However, the Progress "enable" statement along with the "wait-for" statement allows fields to be enabled in more than one frame at a time. Do not do this. Enable fields in only one frame at a time.

Rationale

Desktop 2 assumes that only one frame has enabled fields in it at a time. If more than one frame has enabled fields in it at a time then the Desktop 2 html screens develop framing and focus issues.

Examples

In the code example the "RIGHT" version enables two fields in one frame. The "WRONG" version enables the same fields in two different frames.

Right

Code

```plaintext
define variable x as character no-undo.
define variable y as character no-undo.
enable x y with frame a.
wait-for close of current-window.
```

Wrong

Code

```plaintext```

458
define variable y as character no-undo.

enable x with frame a.
enable y with frame b.

wait-for close of current-window.

See Also

N/A
STD-0294 Modifying screens within an editing block or trigger

Summary

Try to avoid making changes to the screen inside of an editing block or trigger. These changes will not occur in the Desktop 2 html screens in the same manner that they appear in CHUI screens.

Description

In CHUI when a screen is changed in an editing block or trigger (This includes label changes, screen value changes, fields appearing or disappearing or any other screen modification) the user sees the change immediately. In Desktop 2 the change will not appear to the user. It is important to keep this limitation in mind when writing an MFG/PRO program. It is acceptable to change the screen as mentioned above but be aware of the limitations of the HTML screens. If the screen change is critical and must work the same way in HTML then you may need to restructure the CHUI program (see example).

Rationale

With the advent of the Desktop 2 HTML UI it is necessary to think about MFG/PRO UI programming from a different viewpoint. With the HTML UI there is a more limited interaction between the UI and the MFG/PRO code. In CHUI the UI and the code are tightly coupled, in HTML UI they are loosely coupled. The HTML UI is refreshed only after the user submits data or navigates back (The CHUI equivalents of these are “F1” (GO) and “F4” (BACK)). So any other action the user takes on an HTML screen (like tabbing out of a field) will not be seen by an MFG/PRO editing block or trigger and thus no changes will occur to the HTML screen.

Examples

The following WRONG example is taken from qqdp.p where a trigger on one field is used to modify the screen value of another field. The RIGHT example breaks up the one update statement of the WRONG example into two update statements with the result that no trigger is needed.

Right

Code

```plaintext
update eximmode docid with frame a.
if docid <> "0" then do:
   editflname = docid + ".qqd".
   display editflname (editpath + editflname) @ editpath with frame a.
end.
update editflname browse_d with frame a.
```

Wrong

Code

```plaintext
update editflname browse_d with frame a.
```
QAD Development Standards

See Also

N/A
STD-0295 Storage Areas on tables and indexes

Summary

All new schema tables and indexes must be assigned an appropriate storage area in Progress.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0295</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Tables, DB Indexes</td>
</tr>
<tr>
<td>Applicability</td>
<td>MFG/PRO eB and up</td>
</tr>
</tbody>
</table>

Description

All QAD Products:

When submitting new schema tables and indexes to the QAD architects and DBAs for approval, the schema definitions must include an appropriate Progress storage area name assigned to each table and index based on the following criteria.

- The storage area name should be in uppercase.
- The storage area for a new table should be selected from the product's existing storage areas based upon the table's purpose and expected size and growth characteristics.
- Storage areas for indexes should match the storage area for the table on which the index is applied, but the index area will have the suffix "_.IDX" appended to it.

For MFG/PRO (QAD SE and EE) and QAD Financials

In order to find out the storage area on an existing table, developers can use the Unix tool "getscharea", which takes a list of table names as arguments. Please refer to the Unix command "qhelp getscharea" for more details. This tool should be used when adding indexes to existing QAD SE, EE and Financials tables to ensure they are assigned to the correct storage area.

- If a designer feels that a new table warrants its own storage area for performance reasons, please consult with the DBA team to add that new storage area to the product's schema.
- LOB fields should have their own storage area within a database. The area name is prefixed by the name of the database followed by underscore and 'LOB'. For example, the storage area for all LOB fields in the admin database is called "ADMIN_LOB".
- Indexes for the new table will all be assigned to a single separate, new storage area; the storage area for the indexes will have the same name as the storage area for their table, but the index area will have the suffix "_.IDX" appended to it.
- Once new storage areas are approved by project Architects and the QAD DBAs, the Database team will update its tools to make correct use of the new storage areas.

QAD SE Schema Submittal Notes:

1. QAD SE product development databases do not use storage areas. Instead, this information is kept in Development Systems records to be used during release creation. All QAD SE development databases actually contain only one area called "Schema Area".
2. In order to pass the storage area over to the QAD DBAs, the area names should be identified in comments at the top of the Progress .df file which is submitted to Development Systems for approval. Assigning the AREA attribute on tables and indexes is no longer necessary and the submitted .df may be left with the default AREA of "Schema Area" or the AREA attribute may be simply removed. The comment should take the form of the examples in the section below.
QAD EE Schema Submittal Notes:
Since QAD EE development environments use the Build Framework model, private and public EE development environments use the actual storage areas. This differs from QAD SE environments which use only the schema area and lists storage area names in comments at the top of the delta .df file. Therefore, EE developers should specify storage areas for new tables and indexes directly in the data definitions (.df) file produced by their project. This delta .df file - naming the actual storage area within each table and index definition - is what must be submitted to the DBAs for approval prior to committing project deliverables within the Build Framework.

QAD EE Storage Areas:
Beginning with QAD EE 2010, Progress Type II storage areas were made the default storage area format. Type II storage areas allocate space differently than do Type I areas and are designed to provide faster access to data and prevent fragmentation that can occur with Type I areas. Due to these advantages, QAD was able to reduce the number of storage areas in the main database from approximately 40 prior to 2010 EE down to 6, which reduces system administration effort. All new database objects created for EE should use these Type II storage areas.

<table>
<thead>
<tr>
<th>Area</th>
<th>Size</th>
<th>Growth</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIM</td>
<td>varied size</td>
<td>varied growth</td>
<td>All AIM (QAD Warehousing) tables</td>
</tr>
<tr>
<td>AIM_IDX</td>
<td>varied size</td>
<td>varied growth</td>
<td>All AIM (QAD Warehousing) indexes</td>
</tr>
<tr>
<td>FIN</td>
<td>large size</td>
<td>large growth</td>
<td>All Finance tables</td>
</tr>
<tr>
<td>FIN_IDX</td>
<td>large size</td>
<td>large growth</td>
<td>All Finance indexes</td>
</tr>
<tr>
<td>MFG</td>
<td>large size</td>
<td>large growth</td>
<td>All operational tables</td>
</tr>
<tr>
<td>MFG_IDX</td>
<td>large size</td>
<td>large growth</td>
<td>All operational tables</td>
</tr>
</tbody>
</table>

The main EE database:

The Admin database:

<table>
<thead>
<tr>
<th>Area</th>
<th>Size</th>
<th>Growth</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADMIN</td>
<td>medium size</td>
<td>minimal growth</td>
<td>All Admin tables</td>
</tr>
<tr>
<td>ADMIN_IDX</td>
<td>medium size</td>
<td>minimal growth</td>
<td>All Admin indexes</td>
</tr>
<tr>
<td>ADMIN_LOB</td>
<td>large size</td>
<td>medium growth</td>
<td>All Large Objects</td>
</tr>
</tbody>
</table>

The Help database:

<table>
<thead>
<tr>
<th>Area</th>
<th>Size</th>
<th>Growth</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>HELP</td>
<td>medium size</td>
<td>no growth</td>
<td>All Help tables</td>
</tr>
<tr>
<td>HELP_IDX</td>
<td>medium size</td>
<td>no growth</td>
<td>All Help indexes</td>
</tr>
</tbody>
</table>

If a designer feels that a new table warrants its own storage area for performance reasons, please consult with the Database team to add that new storage area to the schema. Indexes for the new table will also all be assigned to a single new storage area; the storage area for the indexes will have the same name as the storage area for their table but with the "_IDX" suffix. Once new storage areas are approved, the Database team will update their tools to make correct use of the new storage areas.

QAD SE (and pre-2010 EE) Storage Areas:

QAD SE (along with QAD EE releases prior to 2010) use Progress Type I storage areas. Type I storage areas are still supported by Progress and can be used alongside, or instead of, Type II areas (such as when a 2008 EE customer upgrades to 2010 EE). QAD segregated SE and pre-2010 EE databases into many storage areas having specific purposes in order to provide optimal performance under the Type I space allocation method. All new database objects created for SE should continue to use these Type I storage areas.
For tables in the MFG/PRO qaddb database:

<table>
<thead>
<tr>
<th>Area</th>
<th>Size</th>
<th>Growth</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTMMPSTR</td>
<td>small</td>
<td>varied</td>
<td>Audit Trail tables</td>
</tr>
<tr>
<td>ATTMMPSTR_IDX</td>
<td>small</td>
<td>varied</td>
<td>Audit Trail indexes</td>
</tr>
<tr>
<td>CONTROL</td>
<td>small</td>
<td>minimal</td>
<td>Various module control tables</td>
</tr>
<tr>
<td>CONTROL_IDX</td>
<td>small</td>
<td>minimal</td>
<td>Various module control indexes</td>
</tr>
<tr>
<td>ESIG</td>
<td>small</td>
<td>varied</td>
<td>E-Signature tables</td>
</tr>
<tr>
<td>ESIG_IDX</td>
<td>small</td>
<td>varied</td>
<td>E-Signature indexes</td>
</tr>
<tr>
<td>GLECDET</td>
<td>small</td>
<td>minimal</td>
<td>GL Entity Consolidation</td>
</tr>
<tr>
<td>GLECDET_IDX</td>
<td>small</td>
<td>minimal</td>
<td>GL Entity Consolidation</td>
</tr>
<tr>
<td>GLRPWRTR</td>
<td>medium</td>
<td>varied</td>
<td>GL Report Writer</td>
</tr>
<tr>
<td>GLRPWRTR_IDX</td>
<td>medium</td>
<td>varied</td>
<td>GL Report Writer</td>
</tr>
<tr>
<td>GLTDET</td>
<td>medium</td>
<td>varied</td>
<td>General Ledger Transaction Detail</td>
</tr>
<tr>
<td>GLTDET_IDX</td>
<td>medium</td>
<td>varied</td>
<td>General Ledger Transaction Detail</td>
</tr>
<tr>
<td>GLTRHIST</td>
<td>large</td>
<td>varied</td>
<td>General Ledger Transaction History</td>
</tr>
<tr>
<td>GLTRHIST_IDX</td>
<td>large</td>
<td>varied</td>
<td>General Ledger Transaction History</td>
</tr>
<tr>
<td>GRS</td>
<td>medium</td>
<td>varied</td>
<td>Requisitions</td>
</tr>
<tr>
<td>GRS_IDX</td>
<td>medium</td>
<td>varied</td>
<td>Requisitions</td>
</tr>
<tr>
<td>HISTORY</td>
<td>large</td>
<td>rapid</td>
<td>Several module archive</td>
</tr>
<tr>
<td>HISTORY_IDX</td>
<td>large</td>
<td>rapid</td>
<td>Several module archive</td>
</tr>
<tr>
<td>INTRASTAT</td>
<td>medium</td>
<td>varied</td>
<td>Import/Export</td>
</tr>
<tr>
<td>INTRASTAT_IDX</td>
<td>medium</td>
<td>varied</td>
<td>Import/Export</td>
</tr>
<tr>
<td>MRPDET</td>
<td>medium</td>
<td>varied</td>
<td>Material Requirements Planning</td>
</tr>
<tr>
<td>MRPDET_IDX</td>
<td>medium</td>
<td>varied</td>
<td>Material Requirements Planning</td>
</tr>
<tr>
<td>QADWKFL</td>
<td>small</td>
<td>varied</td>
<td>QAD Work Table</td>
</tr>
<tr>
<td>QADWKFL_IDX</td>
<td>small</td>
<td>varied</td>
<td>QAD Work Table indexes</td>
</tr>
<tr>
<td>REFERENCE</td>
<td>medium</td>
<td>minimal</td>
<td>System Reference Data</td>
</tr>
<tr>
<td>REFERENCE_IDX</td>
<td>medium</td>
<td>minimal</td>
<td>System Reference Data</td>
</tr>
<tr>
<td>STATIC</td>
<td>small</td>
<td>minimal</td>
<td>QAD master data</td>
</tr>
<tr>
<td>STATIC_IDX</td>
<td>small</td>
<td>minimal</td>
<td>QAD master data</td>
</tr>
<tr>
<td>TRANSACTION</td>
<td>large</td>
<td>rapid</td>
<td>heavy use</td>
</tr>
<tr>
<td>TRANSACTION_IDX</td>
<td>large</td>
<td>rapid</td>
<td>heavy use</td>
</tr>
<tr>
<td>TRGLDET</td>
<td>medium</td>
<td>medium</td>
<td>Inventory Transaction and GL Cross-Ref</td>
</tr>
<tr>
<td>TRGLDET_IDX</td>
<td>medium</td>
<td>medium</td>
<td>Inventory Transaction and GL Cross-Ref</td>
</tr>
<tr>
<td>TRHIST</td>
<td>medium</td>
<td>rapid</td>
<td>Inventory Transaction History</td>
</tr>
<tr>
<td>TRHIST_IDX</td>
<td>medium</td>
<td>rapid</td>
<td>Inventory Transaction History</td>
</tr>
<tr>
<td>WORKFILE</td>
<td>varied</td>
<td>varied</td>
<td>Various module work tables</td>
</tr>
<tr>
<td>WORKFILE_IDX</td>
<td>varied</td>
<td>varied</td>
<td>Various module work tables</td>
</tr>
</tbody>
</table>
For tables in the MFG/PRO qadadm (Admin) database:

<table>
<thead>
<tr>
<th>Area</th>
<th>Size</th>
<th>Growth</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPCONF</td>
<td>medium size</td>
<td>minimal growth</td>
<td>Component Procedure tables</td>
</tr>
<tr>
<td>COMPCONF_IDX</td>
<td>medium size</td>
<td>minimal growth</td>
<td>Component Procedure indexes</td>
</tr>
<tr>
<td>GUI</td>
<td>medium size</td>
<td>minimal growth</td>
<td>All Admin tables</td>
</tr>
<tr>
<td>GUI_IDX</td>
<td>medium size</td>
<td>minimal growth</td>
<td>All Admin indexes</td>
</tr>
<tr>
<td>ADM_LOB</td>
<td>large size</td>
<td>medium growth</td>
<td>All Large Objects</td>
</tr>
</tbody>
</table>

For tables in the MFG/PRO qadadm (Admin) database:

<table>
<thead>
<tr>
<th>Area</th>
<th>Size</th>
<th>Growth</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFGHELP</td>
<td>medium size</td>
<td>no growth</td>
<td>All Help tables</td>
</tr>
<tr>
<td>MFGHELP_IDX</td>
<td>medium size</td>
<td>no growth</td>
<td>All Help indexes</td>
</tr>
</tbody>
</table>

There are some tables in the existing MFG/PRO schema that, in the past, Development Systems had identified as benefiting from having their own storage area, largely for performance reasons. Therefore, when adding indexes to these existing tables, be sure to use the appropriate storage area.

If a designer feels that a new table warrants its own storage area for performance reasons, please consult with the Architecture team to add that new storage area to the schema. Indexes for the new table will all be assigned to a single separate, new storage area; the storage area for the indexes will have the same name as the storage area for their table, but the index area will have the suffix "_IDX" appended to it. Once new storage areas are approved, notify the Database team so they can update their tools to make correct use of the new storage areas.

In order to pass the storage area over to the QAD DBA, the area names should be identified in comments at the top of the Progress .df file which is submitted to Development Systems for approval. Assigning the AREA attribute on tables and indexes is no longer necessary and the submitted .df may be left with the default AREA of "Schema Area" or the AREA attribute may be simply removed. The comment should take the form of the examples in the section below.

Rationale

QAD product databases are created with the tables and indexes allocated into Progress storage areas. The storage areas defined in Progress are also used as the names of Oracle tablespaces for QAD's Oracle customers. Storage areas in Progress allow our customers some measure of control over the physical location of database tables and indexes. The names of storage areas are maintained by Development Systems and a significant amount of work is required to add a new one. It is therefore essential that tables and indexes be assigned to one of the approved storage areas whenever possible, and that the area be appropriate to the type of data being stored.

Examples

The following examples reflect only QAD SE and other products which do not use the Build Framework development model.

The following shows examples of what should appear in QAD SE .df files submitted to Development Systems for schema changes.

Right Design

In the qaddb db:
# Storage Areas

<table>
<thead>
<tr>
<th>Table</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>so_mstr</td>
<td>TRANSACTION</td>
</tr>
<tr>
<td>cm_mstr</td>
<td>STATIC</td>
</tr>
<tr>
<td>poc_ctrl</td>
<td>CONTROL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>so_nbr</td>
<td>TRANSACTION_IDX</td>
</tr>
<tr>
<td>cm_addr</td>
<td>STATIC_IDX</td>
</tr>
<tr>
<td>poc_index1</td>
<td>CONTROL_IDX</td>
</tr>
<tr>
<td>tr_trnbr</td>
<td>TRHIST_IDX</td>
</tr>
</tbody>
</table>

```
ADD TABLE "so_mstr"
```

In the qadadm db:

```
# Storage Areas
# Table            | Area        |
# brw_mstr         | GUI         |
# edc_ctrl         | GUI         |

# Index            | Area        |
# brw_view         | GUI_IDX     |
# edc_index1       | GUI_IDX     |

ADD TABLE "brw_mstr"
```

Wrong Design

In the following .df file example, the commented section for storage areas is missing. In addition, the storage area for cm_mstr is not appropriate.

```
ADD TABLE "so_mstr"
   AREA "Schema Area"
...

ADD TABLE "cm_mstr"
   AREA "CONTROL"
...
```

See Also

N/A
STD-0296 low_date and hi_date global variables - do not store in database

Summary
The value of MFG/PRO's global shared variables low_date and hi_date must never be stored in the database.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0296</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Logic for Dates</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description
The value of MFG/PRO's global shared variables low_date and hi_date must never be stored in the database.
Typically, when we want to suggest the beginning of time or the end of time for date ranges in the database, we'll simply leave the beginning or end date undefined (?).

Background
The global shared variables low_date and hi_date represent the minimum and maximum date values that are expected to be stored in the MFG/PRO database. These variables are intended solely for range checking on date fields, and their primary use is in situations where "From" and "To" Dates are used as selection criteria for a report and the user does not enter a specific date range. The input variables are typically set to low_date and hi_date to allow the retrieval of all dates.

low_date and hi_date are defined in mfdeclire.i and are initialized in mf1a.p for character and gui clients and in mfwb02.p for NetUI clients.

Rationale
Displaying a date field which has been set to hi_date or low_date causes a Progress run time error unless the user's century window has been set to include the date. Most dates within MFG/PRO are formatted to display only the last two digits of the year. The century information is supplied by the value of the Progress -yy parameter. It is expected that the user has carefully selected a -yy setting that captures the range of dates within their production database. Date values that fall outside the century window, hi_date for example, cannot be properly displayed and will be displayed as /???. In addition, the following error message is displayed: "Value 12/31/3999 cannot be displayed using 99/99/99".

Another reason to avoid storing the values of hi_date and low_date in the database is to avoid updating the database if there is ever a need to change the values of these two global variables in the future.

Examples

Right

Code

```define variable to_date like vt_end no-undo. /* to_date is a local variable */
if to_date = ? then to_date = hi_date.
display if to_date <> hi_date then to_date else ? .```

Wrong
Code

```java
if vt_end = ? then vt_end = hi_date. /* vt_end is a schema field */
display vt_end.
```

See Also

N/A
STD-0297 Reuse attributes of common schema fields

Summary

When defining new schema fields that are related to existing common MFG/PRO fields, reuse the attributes of the common fields as outlined in this document.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0297</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>5</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

If a schema field is added to MFG/PRO that is related to an existing field of the common forms listed below, assign the attributes of the new field to be consistent with the common MFG/PRO field. Fields are said to be related if the one field defaults its data from the other, or if data in one field is validated against data in the other.

<table>
<thead>
<tr>
<th>Object</th>
<th>Schema Suffix</th>
<th>Format</th>
<th>MAX-WIDTH</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account</td>
<td>_acct</td>
<td>x(8)</td>
<td>80</td>
<td>&lt;Desc&gt; Account (where &lt;Desc&gt; can be one or more words describing the account acceptable abbreviation is Acct)</td>
</tr>
<tr>
<td>Address ID</td>
<td>_addr</td>
<td>x(8)</td>
<td>80</td>
<td>&lt;varies&gt;</td>
</tr>
<tr>
<td>Comment Key</td>
<td>_cmtindx</td>
<td>&gt;&gt;&gt;&gt;&gt;&gt;9</td>
<td>n/a</td>
<td>Comment Index</td>
</tr>
<tr>
<td>Cost Center</td>
<td>_cc</td>
<td>x(4)</td>
<td>30</td>
<td>&lt;Desc&gt; Cost Ctr (where &lt;Desc&gt; can be one or more words describing the cost center: acceptable abbreviation is CC)</td>
</tr>
<tr>
<td>Currency</td>
<td>_curr</td>
<td>x(3)</td>
<td>30</td>
<td>Currency</td>
</tr>
<tr>
<td>Customer</td>
<td>_cust</td>
<td>x(8)</td>
<td>80</td>
<td>Customer (or Sold-To where appropriate)</td>
</tr>
<tr>
<td>Date</td>
<td>n/a</td>
<td>99/99/99</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Description</td>
<td>_desc</td>
<td>x(24)(can vary)</td>
<td>80</td>
<td>Description</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Length</th>
<th>Format</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doc Amount</td>
<td></td>
<td>n/a</td>
<td>-9.99&lt;</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Domain</td>
<td></td>
<td>_domain</td>
<td>x(8)</td>
<td>8</td>
<td>Domain</td>
</tr>
<tr>
<td>Effective Date</td>
<td>(used only for the date when the transaction will be effective in GL)</td>
<td>_effdate</td>
<td>99/99/99</td>
<td>n/a</td>
<td>Effective</td>
</tr>
<tr>
<td>End User</td>
<td></td>
<td>_eu_nbr</td>
<td>x(8)</td>
<td>80</td>
<td>End User</td>
</tr>
<tr>
<td>Engineer</td>
<td></td>
<td>_eng_code</td>
<td>x(8)</td>
<td>80</td>
<td>Engineer</td>
</tr>
<tr>
<td>Entity</td>
<td></td>
<td>_entity</td>
<td>x(4)</td>
<td>30</td>
<td>Entity</td>
</tr>
<tr>
<td>Exchange Rate</td>
<td></td>
<td>_ex_rate</td>
<td>&gt;&gt;&gt;,&gt;&gt;&gt;&gt;&gt;&gt;,9.9&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;</td>
<td>38</td>
<td>Exch Rate</td>
</tr>
<tr>
<td>Exchange Rate 2</td>
<td></td>
<td>_ex_rate2</td>
<td>&gt;&gt;&gt;,&gt;&gt;&gt;&gt;&gt;&gt;,9.9&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;</td>
<td>38</td>
<td>Exch Rate 2</td>
</tr>
<tr>
<td>Exchange Rate Type</td>
<td></td>
<td>_ex_ratetype</td>
<td>x(8)</td>
<td>80</td>
<td>Rate Type</td>
</tr>
<tr>
<td>Exchange Usage Key</td>
<td></td>
<td>_exru_seq</td>
<td>&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;9</td>
<td>n/a</td>
<td>Sequence</td>
</tr>
<tr>
<td>Fiscal Period</td>
<td></td>
<td>_per</td>
<td>&gt;&gt;9</td>
<td>n/a</td>
<td>Period</td>
</tr>
<tr>
<td>GL Money</td>
<td></td>
<td>n/a</td>
<td>-9.99&lt;</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Item</td>
<td></td>
<td>_part</td>
<td>x(18)</td>
<td>30</td>
<td>Item Name</td>
</tr>
<tr>
<td>Location</td>
<td></td>
<td>_loc</td>
<td>x(8)</td>
<td>80</td>
<td>Location</td>
</tr>
<tr>
<td>Lot/Serial Number</td>
<td></td>
<td>_serial</td>
<td>x(18)</td>
<td>50</td>
<td>Lot/Serial</td>
</tr>
<tr>
<td>OID</td>
<td></td>
<td>n/a</td>
<td>&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;9.9&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;</td>
<td>38</td>
<td>?</td>
</tr>
<tr>
<td>Machine</td>
<td></td>
<td>_mch</td>
<td>x(8)</td>
<td>80</td>
<td>Machine</td>
</tr>
<tr>
<td>Percentage</td>
<td></td>
<td>n/a</td>
<td>-9.9&lt;%</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Phone</td>
<td></td>
<td>n/a</td>
<td>x(16)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Price</td>
<td></td>
<td>_price</td>
<td>-9.99&lt;&lt;</td>
<td>38</td>
<td>Price</td>
</tr>
<tr>
<td>Product Line</td>
<td></td>
<td>_prod_line</td>
<td>x(4)</td>
<td>30</td>
<td>Product Line</td>
</tr>
<tr>
<td>Production Line</td>
<td></td>
<td>_production_line</td>
<td>x(8)</td>
<td>80</td>
<td>Production Line</td>
</tr>
<tr>
<td>Project</td>
<td></td>
<td>_project</td>
<td>x(8)</td>
<td>80</td>
<td>Project</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td>Length</td>
<td>Type</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------</td>
<td>------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>Quantity</td>
<td>n/a</td>
<td></td>
<td></td>
<td>-&gt;&gt;9.999&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;</td>
<td></td>
</tr>
<tr>
<td>Salesperson</td>
<td>_slpsn (extent 4)</td>
<td>x(8)</td>
<td></td>
<td>328</td>
<td></td>
</tr>
<tr>
<td>Service Work Code</td>
<td>_ca_int_type</td>
<td>x(8)</td>
<td></td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>Site</td>
<td>_site</td>
<td>x(8)</td>
<td></td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>Sub-Account</td>
<td>_sub</td>
<td>x(8)</td>
<td></td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>Supplier</td>
<td>_supplier (formerly _vend)</td>
<td>x(8)</td>
<td></td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>Tax Class</td>
<td>_taxc</td>
<td>x(3)</td>
<td></td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Tax Environment</td>
<td>_tax_env</td>
<td>x(16)</td>
<td></td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Tax Included</td>
<td>_tax_in</td>
<td>yes/no</td>
<td></td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>Tax Usage</td>
<td>_tax_usage</td>
<td>x(8)</td>
<td></td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Taxable</td>
<td>_taxable</td>
<td>yes/no</td>
<td></td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>Unit of Measure</td>
<td>_um</td>
<td>x(2)</td>
<td></td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Unit of Measure Conversion</td>
<td>_um_conv</td>
<td>&gt;&gt;&gt;&gt;9.999&lt;&lt;&lt;&lt;&lt;&lt;</td>
<td></td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>User ID</td>
<td>_userid</td>
<td>x(8)</td>
<td></td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>Work Center</td>
<td>_wkctr</td>
<td>x(8)</td>
<td></td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>Year</td>
<td>_year</td>
<td>9999</td>
<td></td>
<td>n/a</td>
<td></td>
</tr>
</tbody>
</table>

The schema suffixes mentioned above are in fact reserved only to be used for the objects identified in the table. For example, a designer can use xx_loc & xx_rept_loc to refer to locations, but he shouldn’t use xx_qty_loc to refer to a quantity.

Make sure you do not exceed Progress’ maximums for the data type of the field in your format statements. For example, the maximum value of an integer is +/- 2,147,483,648 (about 2 billion). You can’t even use billions in the format because it isn’t 9 billion, so you must only allow hundreds of millions. Violations of this are typically found in dealing with currency amounts where it was assumed that only whole units would be needed, such as the customer’s credit limit cm_cr_limit).
Rationale

Consistency is a key element in a relational database. When programming logic attempts to move data between fields that are related, run-time errors may occur if those schema fields are not defined in a consistent manner. In addition, having consistent attributes among related fields helps to identify the relationships among tables when viewing the schema.

Examples

N/A

See Also

STD-0302 Define related fields with similar schema properties
STD-0325 Use OID fields as foreign keys to relate qadcb tables (eB3 and up)
DST-0005 Developing with OID fields (eB3 and up)
STD-0298 Use gpbrparm.i to pass parameters to lookups

Summary

Use gpbrparm.i to pass parameters to lookups.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0298</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Lookups created via Browse Maintenance often need runtime data to provide the proper filter for the query. Passing data from the Progress program to the lookup is limited because the lookup is not run directly from the Progress program but is run via a separate mechanism. Often the lookup will need to filter on a value in one of the MFG/PRO global variables (global_part, global_site etc). This is fine and these can be used directly in the lookup. However, other times the lookup will need data not in any of the global variables (i.e. values of current screen fields or values retrieved from database records). For these cases use gpbrparm.i.

The following list of parameters can be set in the maintenance program via gpbrparm.i. These parameters can be used directly in the query of the lookup.

- Character parameters:
  c-brparm1
  c-brparm2
  c-brparm3
  c-brparm4
  c-brparm5

- Decimal parameters:
  n-brparm1
  n-brparm2
  n-brparm3

- Date parameters:
  d-brparm1
  d-brparm2

- Logical parameter:
  l-brparm1

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

The usage of gpbrparm.i is as follows:

```
{gpbrparm.i &browse=<lookup-name>; &parm=<param-name> &val=<param-value>}
```

Where:

- `<lookup-name>` is the name of the lookup program
- `<param-name>` is the name of one of the parameters listed above
- `<param-value>` is the value you wish to assign to the parameter - this can be a field, variable or literal string.
When using `gpbrparm.i` try to avoid placing the call to it inside of an editing block. Try to place it outside of any editing blocks and before the update/prompt-for/set statements of the field(s) that will be using the lookup. (This is a limitation imposed by the Desktop HTML engine which has difficulty with code in editing blocks). Lookups often need runtime data for filtering and `gpbrparm.i` provides a consistent mechanism, and in some cases the only mechanism, to achieve this.

The right example shows proper code usage of `gpbrparm.i`. The wrong example illustrates misuse of global variables as lookup parameters. It is OK to use global variables in lookups but they should not be forced to take on data that they were not intended for. In these cases `gpbrparm.i` should be used instead.

**Examples**

**Right**

**Code**

```plaintext
/* 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"}
```

**Wrong**

**Code**

Don't force a global variable to be used where `gpbrparm.i` can be used instead.

```plaintext
global_ref = mfguser.
global_addr = "operator".
```

**See Also**

STD-0292 Use `gpbranch.p` to handle branched lookups
STD-0288 Use lookups created via 'Browse Maintenance' - do not use scrolling window lookups
STD-0300 Use gpwait.i to pause for display only frames

Summary

Use gpwait.i to pause for display only frames

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0300</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Desktop UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>MFG/PRO eB and up</td>
</tr>
</tbody>
</table>

Description

In MFG/PRO maintenance programs it is sometimes necessary to have a frame that does only data display and no prompting. Because there is no field being prompted for a pause is needed to hold the frame in view until the user presses the space bar. This pause can occur in several ways - explicitly when the programmer adds the pause statement, automatically when a frame is about to be hidden and 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.

The Desktop engine that retrieves information from the MFG/PRO character screen cannot execute properly when the system is paused (i.e. “Press space bar to continue”). The engine’s code is fired by Progress triggers, however, Progress will not execute triggers when the system is paused. gpwait.i replaces or 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. This occurs only when the Desktop is running and will not occur in a CHUI session.

The known exceptions are “reports and inquiries where the output can be specified”. The reason for this has to do with the output to “Page” option. In MFG/PRO output to “Page” works by sending the report/inquiry output to a flat file and then using a special Progress program to read the file and display it on the screen. An important point here is that Progress does not execute pauses when it is outputting to a file. So as long as the output can be specified the Desktop user can use “Page” and not encounter pauses. Thus it is not necessary to use gpwait.i to handle programs where the output can be specified.

Examples

The RIGHT code shows three examples in which a pause can occur where gpwait should be used. The WRONG code examples show these same situations not using gpwait.i.

Right

Code

DOWN FRAMES
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 follows:

```
for each cm_mstr no-lock:
    display cm_addr with frame a down.
    {gpwait.i &INSIDELOOP=YES &FRAMENAME=a}
end.
{gpwait.i &OUTSIDELOOP=YES}
```
NON DOWN FRAME (about to be hidden)
A frame that has been displayed and is about to be hidden will have an automatic pause that occurs before hiding. In this case do the following:

```plaintext
display cm_addr with frame a.
{gpwait.i}
hide frame a.
```

(See ictriq3.i in eB or eB2 for a real code example)

NON DOWN FRAME (with explicit pause)
A frame that has been displayed and needs an explicit pause after it should be handled as follows. This will insert a "wait-for" when the Desktop is running and a pause statement when CHUI is running.

```plaintext
display cm_addr with frame a.
{gpwait.i &INCLUDEPAUSE=YES}
```

(See gltriq2e.p in eB or eB2 for a real code example)

Wrong

Code

DOWN FRAMES

```plaintext
for each cm_mstr no-lock:
    display cm_addr with frame a down.
    /* An automatic pause will occur here when Progress
     * has drawn the last line in the frame and is about
     * to start at the top of the frame.
     */
end.
```

NON DOWN FRAME (about to be hidden)

```plaintext
display cm_addr with frame a.
hide frame a.
```

NON DOWN FRAME (with explicit pause)

```plaintext
display cm_addr with frame a.
if not batchrun then pause.
```

See Also

N/A
STD-0301 Oracle MAX-WIDTH (SQL-WIDTH) must be specified on new database fields

Summary

The Progress MAX-WIDTH property (SQL-WIDTH in versions prior to OE 10) must be specified on all new fields in the MFG/PRO databases. MAX-WIDTH should also be specified on fields of all other QAD products.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0301</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

MAX-WIDTH is used to produce appropriate Oracle and SQL Server column widths when converting Progress databases to Oracle or SQL Server. Additionally, in Progress databases, MAX-WIDTH specifies the maximum amount of data that SQL clients such as ODBC products can access. SQL clients and ODBC tools can only access the portion of a data value up to the MAX-WIDTH limit. The MAX-WIDTH can be accessed in the OpenEdge 10 Data Dictionary via the “SQL Properties...” option, which is under the Schema menu in CHUI or the Options menu in GUI. In Progress v9, the same attribute is called SQL-WIDTH.

When creating new fields in an application's schema, ensure that the correct MAX-WIDTH is defined based on the following rules. Failure to do so may cause run-time errors if program logic tries to store data that is larger than the defined size of the field's width in an Oracle or SQL Server database, and ODBC clients could fail to find records or will retrieve truncated data values when the actual data length exceeds the defined column width on Progress databases. For more information about Oracle storage requirements, please see standard STD-0090 “Schema field formats should support storage requirements in Oracle” referenced below.

Sample Oracle and SQL Server runtime error:

If an Oracle or SQL Server column definition is not large enough to hold the data, Progress error "A column in this row being inserted or updated is too large (4212)" occurs and the record write fails.

MFG/PRO Usage Notes:

- It is important to properly design the MAX-WIDTH of a field the first time the field is introduced in the MFG/PRO schema. Starting with MFG/PRO eB2.1, QAD is using function-based indexes in the Oracle database. When a field that participates in a function-based index is subsequently changed, the index must now be dropped and recreated, thereby adding additional time to apply schema changes to Oracle. To avoid this, make sure you allow for future growth of a field when determining the proper MAX-WIDTH at design time.
- If the new field being created is one of the required fields found in the "Required Table Fields" standard referenced below, see that standard for the correct MAX-WIDTH to use.
- If the field matches any of the common fields found in the "Reuse attributes of common schema fields" standard referenced below, see that standard for the correct MAX-WIDTH to use.
- For all other fields, use the rules prescribed for All Products below.

All Product Usage Notes:

For character fields:

- If the field is related to another character field already in the schema, use the MAX-WIDTH from the existing field.
- Otherwise, assign the MAX-WIDTH to the number of Progress format characters. For example: for a format of "x(8)", MAX-WIDTH should be 8. Both the MAX-WIDTH and Progress display format should accommodate the largest number of bytes expected to be stored in the field.
For decimal fields:

- If the field is related to another decimal field already in the schema, use the MAX-WIDTH from the existing field.
- Otherwise, determine the number of characters displayed in the Progress format up to and including the decimal, i.e. the result of `index(_format,".")`. For example, the result on a format of ".>>><><><>>.9<<<<<<<<<" is 14. If the result is less than or equal to 10, set the MAX-WIDTH to 20. If the result is greater than 10, set the MAX-WIDTH to 38. The MAX-WIDTH becomes the precision for the decimal field in Oracle and SQL Server and therefore includes 10 decimals places in addition to the result above.

For integer fields:

- Leave MAX-WIDTH set to the Progress default, regardless of the format.

For logical fields:

- Leave MAX-WIDTH set to the Progress default, regardless of the format.

For date fields:

- Leave MAX-WIDTH set to the Progress default, regardless of the format.

For character, decimal and integer array fields:

- If the field is related to another array field that has the same number of extents already in the schema, use the MAX-WIDTH from the existing field.
- Otherwise, calculate the MAX-WIDTH of an array as follows:

\[
\text{TOTAL MAX-WIDTH} = ((\text{<Display format>} \times 2) + 2) \times \text{<Total # of extents>}\]

- For example, a character array of 4 extents, each with a format of "x(8)", would have a MAX-WIDTH of \((8 \times 2 + 2) \times 4\) = 72.
- The actual column width for each extent field on the Oracle or SQL Server database =

\[
\left(\text{<MAX-WIDTH of the Progress array field>} - \text{<total # of extent} \times 2\right) / \text{<total # of extents}}
\]

Using the 4 extent example from above, the width of each extent field on Oracle is: \((72 - (4 \times 2)) / 4 = 16\)
(Notice how the width of each extent field on Oracle or SQL Server ends up being at least double the Progress display format)

**Array Field Warning:**

New array fields should not be added to MFG/PRO schema. Array fields are discouraged in all other QAD products, as well. Information about array fields is included here for use with previously existing array fields only. See QAD Development Standard STD-0333 for additional array field restrictions.

For logical & date array fields:

- Please consult with the Architecture Team if you have a need for these types of arrays.

**Array Field Warning:**

New array fields should not be added to MFG/PRO schema. Array fields are discouraged in all other QAD products, as well. Information about array fields is included here for use with previously existing array fields only. See QAD Development Standard STD-0333 for additional array field restrictions.

A field is considered "related" to another field if the one field gets a default value from or is validated against data in the other field. Please see the standard STD-0302 “Define related fields with similar schema properties” for more information.

Even though this standard only applies to new fields added to the schema, a project team may find the need to modify the MAX-WIDTH of an existing field in the current development baseline. If this is necessary, the
MAX-WIDTH should only be increased, never decreased, because there may be customer data residing in that field that would be large enough to cause errors when dumping and relaoding into new schema.

Rationale

QAD supports running MFG/PRO on both the Progress and Oracle databases. In order to properly define the database fields in Oracle, development teams must determine maximum field sizes then provide a valid and correct MAX-WIDTH. By having the correct field sizes defined, Oracle DBAs are better able to understand the functions and relationships within the MFG/PRO schema. In addition, run-time errors will be avoided if the defined width of the field in Oracle supports the storage requirements of the MFG/PRO program logic.

Correct field width definitions in QAD products also enable support for SQL Server databases should that become necessary.

Correct field widths also enable ODBC tools to access entire field contents and avoid truncation and other problems caused when data length exceeds the defined field width.

See Also

STD-0050 Required Table Fields
STD-0064 Character Field Format length limit
STD-0090 Schema field formats should support storage requirements for Oracle
STD-0297 Reuse attributes of common schema fields
STD-0302 Define related fields with similar schema properties
STD-0333 Database fields with Extents are not supported
STD-0350 Maximum Record (Row) Size Limit
STD-0302 Define related fields with similar schema properties

Summary

When defining new fields that are related to existing schema fields, be sure to make the new schema properties consistent with the existing properties. Name the new field such that it indicates that the two fields are related. The design document for the new functionality should also indicate that the new field is related to at least one existing field. Having consistent attributes among related fields is valuable for all QAD products but is especially important in the MFG/PRO schema.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0302</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>3</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

When defining new fields in any QAD product, the designer should first check if the new field is related to any existing schema fields. A field is considered "related" to another field if the one field gets a default value from or is validated against data in the other field. If so, the new field should have schema properties defined that are consistent with the existing field. Properties include labels, format, MAX-WIDTH(SQL-WIDTH in versions prior to OE10) and decimals. The labels can vary slightly, but must still remain consistent. For example, in MFG/PRO, si_site has a label of "Site", but a new field xx_site might be labeled "Receipt Site".

Related fields should also be named appropriately to indicate their relationship. For example, the new field should use the schema suffix and attributes of its related field. Typically, the name of the field that holds the master data for the new related field should be used as the suffix on each table prefix portion of the field name for all related fields. For example:

- **lc_charge** (Logistics Charge Code) is the master data in lc_mstr
  - **ft lc_charge** (Logistics Charge Code) is a related field in ft_mstr
    - *(Notice the "_lc_charge" suffix on the "ft" table prefix part of the field name to indicate a subordinate relationship with the lc_mstr table.)*
  - **pvo lc_charge** (Logistics Charge Code) is a related field in pvo_mstr
    - *(Notice the "_lc_charge" suffix on the "pvo" table prefix part of the field name to indicate a subordinate relationship with the lc_mstr table.)*

**MFG/PRO Usage:**

If the new field is one of the common MFG/PRO fields listed in standard STD-0297, use the schema suffix and attributes identified in that document. If not one of those common fields, the new related field's name should include the name of the field that holds the master data as shown in the example above.

The design document for new functionality should indicate whether new fields are related to any existing fields. The document should specify that new fields are defaulted from or validated against data in existing fields.

Rationale

Consistent field naming practices permit greater visibility and extraction of the relationships between fields in a product's schema. In addition, related fields which share the same schema properties prevent bugs when joining tables and passing data between related fields.

In the past, MFG/PRO field names and attributes lacked consistency. Some of this inconsistency remains today, a relic of early development activities. It used to be somewhat difficult to be able to extract the relationships between
fields in the MFG/PRO database because of a lack of consistency in their naming. This led to confusion when tables were joined. In addition, several bugs surfaced as a result of passing data between related fields which did not share the same schema properties.

By identifying and appropriately naming related fields in QAD product databases, schema relationships will be clearly represented and confusion and bugs will be minimized going forward.

Examples

Design

ft_lc_charge is validated against data in lc_charge
pvo_lc_charge defaults from data in ft_lc_charge

Code

```plaintext
ADD FIELD "lc_charge" OF "lc_mstr" AS character
  FORMAT "x(8)"
  INITIAL ""
  LABEL "Logistics Charge Code"
  COLUMN-LABEL "Chrg Code"
  MAX-WIDTH 80
  MANDATORY

ADD FIELD "ft_lc_charge" OF "ft_mstr" AS character
  FORMAT "x(8)"
  INITIAL ""
  LABEL "Logistics Charge Code"
  COLUMN-LABEL "Chrg Code"
  MAX-WIDTH 80
  MANDATORY

ADD FIELD "pvo_lc_charge" OF "pvo_mstr" AS character
  FORMAT "x(8)"
  INITIAL ""
  LABEL "Logistics Charge Code"
  COLUMN-LABEL "Chrg Code"
  MAX-WIDTH 80
  MANDATORY
```

Wrong

Design

(no indication of relationship in design)

Code
ADD FIELD "lc_charge" OF "lc_mstr" AS character
FORMAT "x(8)"
INITIAL ""
LABEL "Logistics Charge Code"
COLUMN-LABEL "Chrg Code"
MAX-WIDTH 80
MANDATORY

ADD FIELD "ft_lc_charge" OF "ft_mstr" AS character
FORMAT "x(20)" <<---- inconsistent format
INITIAL ""
LABEL "Logistics Charge Code"
COLUMN-LABEL "Chrg Code"
MAX-WIDTH 30 <<---- inconsistent MAX-WIDTH
MANDATORY

ADD FIELD "pvo_charge" OF "pvo_mstr" AS character <<---- inconsistent name
FORMAT "x(8)"
INITIAL ""
LABEL "Charge Code" <<---- inconsistent label
COLUMN-LABEL "Charge" <<---- inconsistent label
MAX-WIDTH 80
MANDATORY

See Also

STD-0297 Reuse attributes of common schema fields
STD-0325 Use OID fields as foreign keys to relate qadb tables (eB3 and up)
DST-0005 Developing with OID fields (eB3 and up)
STD-0303 Variables used to prompt for options

Summary

When prompting a user for options in an application, use a logical data type variable only when the options are opposites of each other. In all other cases, use a character data type variable for the option. An integer data type is allowed, but not recommended.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0303</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Data Handling Logic, DB Fields, Variable Definition</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Many data fields in MFG/PRO are designed to present a number of options that the user can choose from, which indicate certain business processes. When the designer chooses a data type to hold these options, the following guidelines should be used to provide the most flexible and usable solution for the product and its user interface.

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. Generally, logical fields do not carry next/previous logic or lookups on the user interface, and therefore their values must be easily distinguishable from each other. In addition, opposites will always have only two values to choose from and no more options will present themselves in the future.

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.

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. 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 MFG/PRO 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 MFG/PRO customer, not the QAD designer and hence usually do not carry any specific logic per option.

In general, avoid using logical and integer data type fields for application options when the allowable values may be hard to determine or confusing to the MFG/PRO user.

Rationale

Character data type variables provide the most flexibility when it comes to presenting options to an MFG/PRO user. They can easily be extended to accommodate more options in the future. They can also be linked to language details to provide translated options and lookup browsers. Logical data type variables are convenient to use in the code, but do have some limits in their extensibility and user interface which should be considered when designing for options.

Example

Right

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"

Wrong

Variable addressType defined as logical with a format of "Customer/Supplier".

These may be the only two options now, but we may add more supported address types in the future.

Also, Customer and Supplier are not necessarily opposites of each other and the user may not know what options he has to choose from.

Variable billingLevel defined as integer which accepts 1 for detail and 2 for summary.

It may be unclear to a user what options 1 and 2 really mean to the functionality.

See Also

STD-0020 Using Language Detail
STD-0044 Definition And Acceptable Values For Logical Variables In Progress
STD-0304 All CM-controlled files should have copyright & versioning

Summary

Any file that is CM-controlled should have copyright information and versioning.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0304</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>All</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Any file that is controlled by a configuration management tool within QAD should include a header of comments at the top of the file similar to the header required on all source code programs per standard STD-0196. Such files may include ASCII text files, XML documents, etc. The file's header must include QAD copyright information, a title & description of the file and a section on revision history.

The copyright information needs to appear exactly as follows:

```
# Copyright 1986-<Year>; QAD Inc., Carpinteria, CA, USA
# All rights reserved worldwide. This is an unpublished work.
```

where <Year> is substituted with the current year.

The revision history may contain all previous versions of the file, or simply the current version. In addition to the version from the CM tool, the history should include the full name of the developer and the date last modified.

QAD copyright information protects the intellectual property of the company. As more and more of our program logic is moved away from hard code and into data-driven repositories, it becomes important to copyright the data files as well as the program logic that is developed. In addition, the ability to version our data files will better equip our Global Support organization in the analysis and resolution of any software issues related to those files.

Example

```
# ItemMaintPres.xml - XML User Interface Definition for Project X Item Maintenance
#
# Copyright 1986-2002 QAD Inc., Carpinteria, CA, USA
# All rights reserved worldwide. This is an unpublished work.
#
# This file provides the XML definition for the presentation of the Item Maintenance
# function in the Project X user interface.
#
# version: 1.0   By: Joe Developer   Date: 04/13/02
```

See Also

STD-0196 Source file formatting - Header (pre-eB2)
STD-0227 Maximum acceptable Source Line Length - 80 characters
STD-0316 Source file formatting - Header on files in Continus CM (eB2 and up)
STD-0312 Source file formatting - Header on files in RCS (eB2 and up)
STD-0305 Extending frozen schema tables to provide extra QAD fields (eB2 and up)

Summary

These are the rules for extending existing tables in frozen MFG/PRO schema, in order to provide extra fields for QAD development.

This standard applies to mature, frozen MFG/PRO releases of eB2, eB2.1 / QAD SE and QAD EE only. This standard does not apply to other QAD products.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0305</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>3</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Conversions, DB Tables, Table Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities for MFG/PRO versions eB2 and up</td>
</tr>
</tbody>
</table>

Description

The table qtbl_ext in MFG/PRO provides a generic structure for QAD programs to store and manipulate data for which there is no storage available in a frozen schema. The qtbl_ext table contains generically defined keys and fields that handle all data types.

Use of qtbl_ext is restricted to special circumstances! This solution should be used on an exception basis for closed baselines only. This should not be used in current development baselines which are still open to schema changes. Basically, this method should be used by our maintenance organization only to fix previous releases and not by developers creating new functionality for a future release.

The main purpose of the qtbl_ext is to provide a flexible data structure in the event new fields are needed in a frozen MFG/PRO schema table. For example, if a bug is uncovered in a previously released version of MFG/PRO and the solution is designed requiring additional data to be stored in an existing table having insufficient __qad (QAD-reserved) fields, then it is valid to use qtbl_ext to maintain the additional data.

If the frozen schema table contains spare QAD-reserved fields that are available to use, these should be used first before creating additional records in the qtbl_ext table.

Follow this standard and use qtbl_ext when a previous MFG/PRO release needs fields added to an existing table to implement an ECO or retrofit a feature from a subsequent release.

Using qtbl_ext requires additional planning and coding to manage the entries stored in the table. (Refer to the section on rules below.) It is mandatory that the use of qtbl_ext be documented as part of the resolution within the ECO used to resolve the initial coding problem or provide the enhancement.

The ECO should also contain the proper solution, as if the schema was not frozen. This includes the field names, indexes and navigation code required to support the proper solution. Any special processing should be noted in order to assist the task of changing from qtbl_ext to the properly named fields. Since adding new fields to the schema requires architectural approval, the use of qtbl_ext will consequently be reviewed as well.
When the ECO is migrated to a release that accepts schema changes, then the use of qtbl_ext must cease. The designer and developer of the ECO must ensure new releases don't go out with code dependent upon qtbl_ext records. The proper solution is to modify the schema with properly named fields and code the solution to maintain these new fields. It is the responsibility of both the designer and developer on the ECO to make the necessary schema changes. The following tasks are required:

- Submit schema changes to a QAD DBA for review and approval
- Modify programs to refer to the properly named fields
- Create conversion programs to move the data from qtbl_ext (coming from a previous release) and populate the newly added fields.

The qtbl_ext table is reserved for QAD use and it is not to be used by customers to store customer-specific data. The usrw_wkfl is a customer-reserved workfile table for the customers to utilize for their purposes.

**Rules:**

The following are rules governing the use of qtbl_ext:

**Unique Key**

The unique key of the qtbl_ext table is a compound key of several identifier fields and generic character fields. Following is a list of the fields and their intended use:

<table>
<thead>
<tr>
<th>Field</th>
<th>Intended Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>qtbl_owner</td>
<td>The development group that owns the data in the record</td>
</tr>
<tr>
<td>qtbl_table</td>
<td>The name of the existing table (or owning table) being extended</td>
</tr>
<tr>
<td>qtbl_reference</td>
<td>Free-form identifier for what type of data is stored in the record</td>
</tr>
<tr>
<td>qtbl_key1-6</td>
<td>Six free-form character fields to hold the unique key of the owning table</td>
</tr>
</tbody>
</table>

The owner, table, reference and key1 fields must be supplied in order to create records in qtbl_ext. The free-form qtbl_reference field can be anything that identifies what data will be stored in the spare fields provided, such as the project name or number, or an identifier for the functionality being added to the table. The six free-form character fields, qtbl_key1-6 are used to store the values of all fields making up the unique index of the owning table. In most cases, not all six fields will be needed. In fact, starting in eB3 / QAD EE, every qaddb table contains a unique index consisting of only the main OID field so only the qtbl_key1 field will be needed to extend any table. If any of the fields in the unique index are not character fields, they must be converted appropriately when maintaining the qtbl_ext record before and after using the include files described in the Record Maintenance section below.

**Available Extra Fields**

The qtbl_ext table contains 14 fields that can be used to extend an MFG/PRO table: 5 character, 5 decimal, 2 logical, and 2 date datatypes. If more fields of a particular type are needed than exist in one qtbl_ext record, the qtbl_reference data can include a sequence number to identify linked extension records.

For example, in eB2.1 the Automotive Development Group would like to add additional fields to the so_mstr table for the Customer Consignment project. They need to add a character field and three logicals. The records created for a sales order with an ID of SO123 might contain the following data:

```
qtbl_owner = "ADG"          qtbl_owner = "ADG"
qtbl_table = "so_mstr"      qtbl_table = "so_mstr"
qtbl_reference = "CustConsign1"  qtbl_reference = "CustConsign2"
qtbl_key1 = "Domain-Code"   qtbl_key1 = "Domain-code"
qtbl_key2 = "SO123"         qtbl_key2 = "SO123"
qtbl_qadc01 = "data"        qtbl_qad101 = no
qtbl_qad101 = yes           qtbl_qad102 = yes
```

Since the fields are generic, there is a possibility that two different programs can construct the same key to represent different data. This conflict could result in data corruption and undesired results. It is important that thorough research is done before designing the key – this includes checking every target release that will create qtbl_ext entries and not just the initial release.
**Record Maintenance - Create, Read, Update, Delete**

Maintenance should be performed on the qtbl_ext records only through the standard include files provided below. Please review the comments in the include file source code for an explanation of their parameters. Local variables should be defined in the programs using these include files with names that will match the schema fields intended to be added in a future release. These local variables should be used as the parameters to hold the data contained in the spare QAD fields of the qtbl_ext record.

- **gpextget.i** – used to retrieve fields in a qtbl_ext record when the owning record is read
- **gpextset.i** – used to update fields in an existing qtbl_ext record when changes to the owning record are committed (may also create or delete the qtbl_ext record when necessary)
- **gpextdel.i** – used to delete a qtbl_ext record when the owning record is deleted

Programs should be designed so that qtbl_ext records are only created when non-blank or non-null data is stored in the record. Furthermore, qtbl_ext records should be deleted if all the values in the record have changed from non-blank to blank. (The general include file gpextset.i will handle most of this.) Finally, qtbl_ext records should be deleted when the associated owning table record is deleted, or archived when the owning table record is archived.

**Convert to Properly Named Fields**

When qtbl_ext is used, it is the responsibility of the designer and software engineer to create properly named fields in the next release of MFG/PRO where schema updates are accepted. It is also their responsibility to modify the programs to utilize the properly named fields instead of the qtbl_ext fields. Finally, it is the responsibility of the designer and software engineer to develop a conversion program to move the values from qtbl_ext to the properly named fields in the owning table. The conversion includes deleting qtbl_ext records after the data has been moved. This ensures qtbl_ext does not contain extraneous or duplicate information, and that the size of qtbl_ext is reduced.

**Reference**

The attached slide illustrates how qtbl_ext and qad_wkfl are used in frozen schema starting with MFG/PRO eB2 and beyond. The illustration also shows how those table references should be replaced in the next development baseline once the schema is open.

Download "Extending Frozen Schema" PowerPoint

**Rationale**

The qtbl_ext table allows R&D and Maintenance to expand the tables used during a period of time where the schema cannot be modified. The use of qtbl_ext must be controlled and this standard provides the rules by which that control can be established.

**Examples**

A need arises for the EIDG to store a second bank code in A/R, along with a flag indicating whether this second bank is to be used instead of the first. The new fields should be maintained as part of the Accounts Receivable master table (ar_mstr), but the eB3 / QAD EE schema is frozen and no additional fields can be added at this time. The qtbl_ext table is used to store the new bank code and new flag until the properly named fields are added to the schema in some future release.

The programs below show how to maintain the second bank code and flag in a maintenance screen, and store the values in a qtbl_ext record. The statements highlighted in **GREEN** are additions to the standard program.

---

The OWNER, TABLENAMESPACE and REFERENCE parameters to the new include files must be surrounded by single quotes (" ") and cannot contain any blanks.

The difference between the coding examples below is that the **WRONG** program creates or maintains a qtbl_ext record **regardless** if there are entries in the ar_bank2 and ar_use_bank2 fields. This will generate qtbl_ext records that only contain the key values, and are "basically" empty. A popular transaction like SO Maintenance could
potentially create thousands of empty qtbl_ext records. In addition, the wrong program uses the qtbl_ext fields directly and does not make use of the standard include files for accessing qtbl_ext. The WRONG example also uses multiple key fields when only one is needed.

The **RIGHT** program has several advantages. First, it detects if there is an entry in any qtbl_ext field and will not create a qtbl_ext record if all fields are empty. Secondly, it deletes the existing qtbl_ext record if all fields in the record are now empty. This ensures the number of qtbl_ext records is as small as possible. Lastly, the implementation hides as much of the qtbl_ext access as possible, which will simplify the replacement of the qtbl_ext fields when the schema is updated in a future release. The WRONG example also uses a single key field since a unique OID field is present in ar_mstr.

**Right**

/* THE NEXT 2 LINES WILL BE REMOVED WHEN ar_bank2 AND */
/* ar_use_bank2 ARE IN THE SCHEMA. */
define variable ar_bank2 as character format "x(2)" no-undo.
define variable ar_use_bank2 like mfc_logical no-undo.

form
ar_bank colon 20
ar_bank2 colon 20
ar_use_bank2 colon 20
with frame bank width 80.

/* THE NEXT SECTION WILL BE REMOVED WHEN ar_bank2 AND */
/* ar_use_bank2 ARE IN THE SCHEMA. */
assign
ar_bank2 = ""
ar_use_bank2 = no.
/* READ */
{gpextget.i &OWNER = 'EIDG'
 &TABLENAME = 'ar_mstr'
 &REFERENCE = 'BankCode2'
 &KEY1 = string(oid_ar_mstr)
 &CHAR1 = ar_bank2
 &LOG1 = ar_use_bank2}

update
ar_bank
ar_bank2
ar_use_bank2
with frame bank.

if lastkey = keycode("F5") or lastkey = keycode("CTRL-D")
then do:

/* THE NEXT SECTION WILL BE REMOVED WHEN ar_bank2 AND */
/* ar_use_bank2 ARE IN THE SCHEMA. */
/* DELETE */
{gpextdel.i &OWNER = 'EIDG'
 &TABLENAME = 'ar_mstr'
 &REFERENCE = 'BankCode2'
 &KEY1 = string(oid_ar_mstr)}
delete ar_mstr
next mainloop.
end.

/* THE NEXT SECTION WILL BE REMOVED WHEN ar_bank2 AND */
/* ar_use_bank2 ARE IN THE SCHEMA. */
/* UPDATE */
{gpextset.i &OWNER = 'EIDG'
 &TABLENAME = 'ar_mstr'
 &REFERENCE = 'BankCode2'
 &KEY1 = string(oid_ar_mstr)
 &CHAR1 = ar_bank2
 &LOG1 = ar_use_bank2}

**Wrong**
define variable ar_bank2 as character format "x(2)" no-undo.
define variable ar_use_bank2 like mfc_logical no-undo.

form
ar_bank colon 20
ar_bank2 colon 20
ar_use_bank2 colon 20
with frame bank width 80.

for first qtbl_ext
where qtbl_table = "ar_mstr"
and qtbl_key1 = ar_domain
and qtbl_key2 = ar_nbr exclusive-lock:
assign
ar_bank2 = qtbl_qadc01
ar_use_bank2 = qtbl_qadl01.
end.

update
ar_bank
ar_bank2
ar_use_bank2
with frame bank.

/* *** THE LOGIC BELOW WILL ALWAYS CREATE A qtbl_ext RECORD *** */
/* *** FOR EACH ar_mstr RECORD, REGARDLESS OF THE VALUES OF *** */
/* *** OF ar_bank2 AND ar_use_bank2. THIS CAUSES MANY *** */
/* *** "BLANK" RECORDS. *** */
if available qtbl_ext then
/* UPDATE */
assign
qtbl_qadc01 = ar_bank2
qtbl_qadl01 = ar_use_bank2.
else do:
/* CREATE */
create qtbl_ext.
assign
qtbl_table = "ar_mstr"
qtbl_key1 = ar_domain
qtbl_key2 = ar_nbr
qtbl_qadc01 = ar_bank2
qtbl_qadl01 = ar_use_bank2.
end.

See Also

STD-0041 Use of user fields in MFG-PRO is NOT allowed
STD-0307 Providing for Extra QAD Fields
STD-0308 Proper Use of qad_wkfl (eB2 and up)
STD-0327 Every table must have a unique index & a primary index (eB3 and up)
STD-0306 Searching for field, table or index usage in MFG-PRO

Summary

When searching for the use of field, table or index names in MFG/PRO, conversion source code and devmstr data should be checked in addition to the MFG/PRO source code itself.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0306</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields, DB Indexes, DB Tables, Devmstr, Environment</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Developers need to search for any use of a field, table or index name if that name needs to be modified or obsoleted in an MFG/PRO release. Typically, this search only included the development source code directories for MFG/PRO. However, in order to find all occurrences of a schema name, the conversion target source code and the devmstr release data must also be examined.

A Unix ‘grep’ or the Development Systems tool ‘biggrep’ on the following file patterns and directories should yield a comprehensive search for a field, table or index name in a particular MFG/PRO release. Before executing these, use the Development Systems tool ‘go’ to establish the working MFG/PRO release. More information on Development Systems tools can be found using the ‘qhelp’ command.

```
*.[iptvw] in $DEVSRC
*.[iptvw] in $STGSRV
tv*.[iptvw] in $STGOBJ/conv/us/src
*.d in $STGSRV/mfg
*.d in $STGSRV/admin (for MFG/PRO versions prior to eB, this is $STGSRV/gui and $STGSRV/cfg)
```

For more information regarding Devmstr-controlled data in the *.d files above, please refer to the “What is Devmstr and what are its uses” document in the Development Systems FAQ Notes database. If a schema name is found in the above search, the name must be corrected or removed from the source code or data (via the Devmstr system if that applies) before the schema name can be modified or obsoleted.

Before changing or obsoleting names used in MFG/PRO schema, it is important to find and/or modify all occurrences of the name in order to avoid compile or run-time errors.
STD-0307 Providing for Extra QAD Fields

Summary

Guideline for providing extra QAD-reserved fields in new and existing tables, including their allowed schema attributes.

Though originally written for MFG/PRO versions prior to eB2, this standard is valid for all QAD products including MFG/PRO versions eB2 and newer (eB2.1, QAD SE, QAD EE, ...)

ID | STD-0307
---|---
Version | 2
Language | Progress
Status | Published
Categories | DB Fields
Applicability | All development activities for MFG/PRO versions before eB2

Description

In an effort to make the enhancement of QAD products easier for development during periods where the schema cannot be modified, QAD-reserved fields should be provided in schema tables. In the STD-0050 Required Table Fields standard (“See Also” list, below) there are a number of fields identified as mandatory. If extra fields are desired due to a high likelihood of the need for enhancement, additional QAD-reserved fields may be added to a table following the allowed schema attributes in the grid which follows.

Guideline for adding additional QAD fields:

Each __qad field (notice the double underscore) will contain a numeric value (for example 01 or 02, ...). The reason for this is to allow for more __qad fields than the required fields. If the table has little growth potential, then the mandatory fields are sufficient. For example the Alternate Unit of Measure Master Table (um_mstr) in MFG/PRO is considered a low-growth table. However, if the table has an average growth potential, it might be a good idea to have 5 QAD-reserved character fields and 2 fields of each other type; decimal, integer, logical and date. Control files would be a good example of this. If the table has a large growth potential, it might be a good idea to have 10 QAD character fields and 3 fields of each other type; decimal, integer, logical and date. For example, the MFG/PRO Customer Master Table (cm_mstr) has average growth potential. This is strictly a judgment call based on the understanding of the use of the table. The goal is to provide for probable future table expansion, while trying to cut back on the creation of excessive fields that will not be used.

Schema Attributes:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Format</th>
<th>MAX-WIDTH</th>
<th>Label</th>
<th>Column Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx__qadc03</td>
<td>Character x(24)</td>
<td>80</td>
<td>?</td>
<td>?</td>
<td>QAD reserved field. <strong>Note:</strong> the numbering has started at 03 because two user QAD fields xx__qadc01 and xx__qadc02 are mandatory.</td>
</tr>
<tr>
<td>xx__qadd01</td>
<td>Decimal -&gt;&gt;&gt;&gt;&gt;&gt;,&gt;&gt;&gt;&gt;&gt;,&gt;&gt;&gt;&gt;&gt;.9&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;&lt;</td>
<td>38</td>
<td>?</td>
<td>?</td>
<td>QAD reserved field</td>
</tr>
<tr>
<td>xx__qad01</td>
<td>Integer -&gt;&gt;&gt;&gt;&gt;&gt;&gt;9</td>
<td>n/a</td>
<td>?</td>
<td>?</td>
<td>QAD reserved field</td>
</tr>
<tr>
<td>xx__qad01</td>
<td>Logical yes/no</td>
<td>n/a</td>
<td>?</td>
<td>?</td>
<td>QAD reserved field</td>
</tr>
<tr>
<td>xx__qadt01</td>
<td>Date 99/99/99</td>
<td>n/a</td>
<td>?</td>
<td>?</td>
<td>QAD reserved field</td>
</tr>
</tbody>
</table>
Where:

- **xx** is the prefix from the table name. (For example: xx = gl where table name = gl_mstr)

- **c** stands for the variable type character
- **d** stands for the variable type decimal
- **i** stands for the variable type integer
- **l** stands for the variable type logical
- **t** stands for the variable type date

**Rationale**

QAD-reserved fields allow R&D and Maintenance personnel to expand the tables used during a period of time where the schema cannot be modified.

**Example**

```sql
ADD FIELD "crm__qadc03" OF "crm_mstr" AS character
  DESCRIPTION "QAD reserved field"
  FORMAT "x(24)"
  INITIAL ""
  MANDATORY
  MAX-WIDTH 80

ADD FIELD "crm__qadd01" OF "crm_mstr" AS decimal
  DESCRIPTION "QAD reserved field"
  FORMAT "->>>>,>>>>,>>>>9.9<<<<<<<<<"
  DECIMALS 10
  INITIAL "0"
  MANDATORY
  MAX-WIDTH 38

ADD FIELD "crm__qadi01" OF "crm_mstr" AS integer
  DESCRIPTION "QAD reserved field"
  FORMAT "->>>>>>>9"
  INITIAL "0"
  MANDATORY
  MAX-WIDTH 4

ADD FIELD "crm__qadl01" OF "crm_mstr" AS logical
  DESCRIPTION "QAD reserved field"
  FORMAT "yes/no"
  INITIAL "no"
  MANDATORY
  MAX-WIDTH 1

ADD FIELD "crm__qadt01" OF "crm_mstr" AS date
  DESCRIPTION "QAD reserved field"
  FORMAT "99/99/99"
  INITIAL ?
  MAX-WIDTH 4
```

**See Also**

- STD-0006 Allowed number of fields per table
- STD-0050 Required Table Fields
- STD-0051 Providing for Extra Customer Fields
- STD-0305 Extending frozen schema tables to provide extra QAD fields (eB2 and up)
- STD-0333 Database fields with Extents are not supported
STD-0308 Proper Use of qad_wkfl (eB2 and up)

Summary

Rules governing the proper use of the qad_wkfl table.

This standard applies to mature, frozen MFG/PRO releases of eB2, eB2.1 / QAD SE and QAD EE only. This standard does not apply to other QAD products.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0308</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Data Handling Logic, Field Access &amp; Storage, Table Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities for MFG/PRO versions eB2 and up</td>
</tr>
</tbody>
</table>

Description

The qad_wkfl is a table in MFG/PRO that provides a generic structure for QAD programs to store and manipulate data for which there is no storage available in a frozen schema. The table contains generically defined keys, fields, and arrays that handle all data types. The qad_wkfl also contains multiple indices for sort and break options.

Use of the qad_wkfl is restricted to special circumstances!

This solution should be used on an exception basis for closed baselines only. This should not be used in current development baselines which are still open to schema changes. Basically, this method should be used by our maintenance organization only to fix previous releases and not by developers creating new functionality for a future release.

The main purpose of the qad_wkfl is to provide a flexible data structure in the event a new structure is needed in a frozen schema. For example, if a project needs bolt-on functionality in a previously released version of MFG/PRO and the solution is designed requiring additional data to be stored in a new table, then it is valid to use the qad_wkfl to maintain the additional data. If the design requires adding additional fields to an existing table, please refer to STD-0305 Extending Frozen Schema Tables to Provide Extra QAD Fields (eB2 and up) for how to use qtbl_ext to accomplish this.

Follow this standard and use qad_wkfl when a previous MFG/PRO release needs a new table to implement an ECO or retrofit a feature from a subsequent release.

Using the qad_wkfl requires additional planning and coding to manage the entries of the qad_wkfl. (Refer to the section on rules below.) It is mandatory that the use of the qad_wkfl be documented as part of the resolution within the ECO used to resolve the initial coding problem.

The ECO should also contain the proper solution, as if the schema was not frozen. This includes the field names, indexes and navigation code required to support the proper solution. Any special processing should be noted in order to assist the task of changing from qad_wkfl to the properly named tables.
When the ECO is migrated to a release that accepts schema changes, then the use of the qad_wkfl must cease. The designer and developer of the ECO must ensure new releases don't go out with code dependent upon qad_wkfl records. The proper solution is to modify the schema with the properly named table and code the solution to maintain these new fields. It is the responsibility of both the designer and developer on the ECO to make the necessary schema changes. The following tasks are required:

- Submit schema changes to QAD DBA for review and approval
- Modify programs to refer to the properly named table
- Create conversion program to move the data from the qad_wkfl (coming from a previous release) and populate the newly added table.

The qad_wkfl should not be used to store records between processing of sub-programs. Instead, temp-tables along with the buffer parameter are the standard way of passing data between programs.

In many cases, the use of bolt-on databases to hold new tables in previous versions of MFG/PRO will eliminate the need to use qad_wkfl. All design specifications will be inspected to ensure proper use of the qad_wkfl.

The qad_wkfl is reserved for QAD use and it is not to be used by customers to store customer-specific data. The usrw_wkfl is a customer-reserved workfile table for the customers to utilize for their purposes.

**Rules:**

The following are rules governing the use of the qad_wkfl:

**Unique Key**

The qad_wkfl's unique key is a compound key of three character fields (qad_key1, qad_key2 & qad_domain). The qad_domain field is the field used to store the domain code to which the data belongs. Since the qad_key1 and qad_key2 fields are generic, there is a possibility that two different programs can construct the same key to represent different data. This conflict could result in data corruption and undesired results. It is important that thorough research is done before designing the key, this includes checking every target release that will create qad_wkfl entries and not just the initial release.

- For eB2 and eB2.1, a grep on the MFG/PRO dev, test and staged source code should yield every use of qad_wkfl records.
- For 93 / QAD EE, which uses the Build Framework development model, use the fwbgrep or fwdevgrep scripts in your private work area or your team's public work area to search for instances of qad_wkfl record usage.

The recommended approach for creating a unique key is to use the name of the new table as qad_key1, and the value of the unique key of the new table as qad_key2. This ensures uniqueness and provides a quick indication of the source of the data. For example, a qad_wkfl entry for a new shipping table would look like the following:

```sql
for first qad_wkfl
    where qad_domain = global_domain
    and qad_key1 = "ship_mstr"
    and qad_key2 = ship_nbr no-lock: end.
```

The qad_wkfl now acts like the new table.

**Convert to Properly Named Table**

When the qad_wkfl is used in a frozen schema scenario, it is the responsibility of the designer and software engineer to create a properly named table in the next release of MFG/PRO where schema updates are accepted. It is also their responsibility to modify the programs to utilize the properly named table instead of the qad_wkfl. Finally, it is the responsibility of the designer and software engineer to develop a conversion program to move the values from the qad_wkfl to the properly named table. The conversion includes deleting qad_wkfl records after the data has been moved. This ensures the qad_wkfl does not contain extraneous, or duplicate, information and the size of the qad_wkfl is reduced.

**Reference**
The attached slide illustrates how qad_wkfl and qtbl_ext are used in frozen schema starting with MFG/PRO eB2 and beyond. The illustration also shows how those table references should be replaced in the next development baseline once the schema is open.

Extending Frozen Schema.ppt.tg.zip

Rationale

The qad_wkfl is a flexible means of storing all types of data in the database, but is not meant to manage data permanently. A lot of the data found in the qad_wkfl today are the result of fixes to MFG/PRO where additional data needs to be stored. This data was never "moved" to its proper table when the next release allowed schema changes to add the target table. As a result, the qad_wkfl is filled with data belonging to various tables. This use of the qad_wkfl needs to be controlled. This standard defines the proper introduction and management of the qad_wkfl.

Examples

A need arises to add a new shipping table to store additional data specific to Sales Order Shipments. The data should be stored in a new table (ship_mstr), but the schema is frozen and no additional tables can be added at this time. The qad_wkfl is used to store the new shipping data until the properly named table is added to the schema in some future release.

The programs below display how to maintain the new table in a maintenance screen, and store the data in a qad_wkfl record.

Right

Code
define input parameter shipnbr as character no-undo.

/* THE TEMP-TABLE WILL BE REMOVED WHEN ship_mstr IS IN THE DB */
define temp-table ship_mstr no-undo
  field ship_domain as character format "x(8)"
  field ship_nbr as character format "x(8)"
  field ship_date as date
  field ship_destination as character format "x(20)"
index ship_nbr is unique primary ship_domain ship_nbr.

form
  ship_date colon 20
  ship_destination colon 20
with frame shipper width 80.

for first ship_mstr
  where   ship_domain = global_domain
          and ship_nbr = shipnbr exclusive-lock: end.
if not available ship_mstr then do:
  create ship_mstr.
  assign   ship_domain = global_domain
          ship_nbr = shipnbr.
end.

/* THE NEXT SECTION WILL BE REMOVED WHEN ship_mstr IS IN THE DB */
for first qad_wkfl
  where qad_domain = global_domain
        and qad_key1 = "ship_mstr"
        and qad_key2 = ship_nbr exclusive-lock:
    assign
          ship_date = qad_datefld[1]
          ship_destination = qad_charfld[1].
end.

update
  ship_date
  ship_destination
with frame shipper.

/* THE NEXT SECTION WILL BE REMOVED WHEN ship_mstr IS IN THE DB */
if available qad_wkfl then
  /* UPDATE */
  assign
    qad_datefld[1] = ship_date
else do:
  /* CREATE */
  create qad_wkfl.
  assign
    qad_domain = global_domain
    qad_key1 = "ship_mstr"
    qad_key2 = ship_nbr
    qad_datefld[1] = ship_date
end.

Wrong

Code

Using qad_wkfl directly in logic is wrong because it increases the overhead associated with removing qad_wkfl in later releases.
define input parameter shipnbr as character no-undo.

form
qad_datefld[1]  colon 20
qad_charfld[1]  colon 20 format "x(20)"
with frame shipper width 80.

for first qad_wkfl
  where qad_domain = global_domain
  and qad_key1 = "ship_mstr"
  and qad_key2 = shipnbr exclusive-lock: end.
if not available qad_wkfl then do:
  create qad_wkfl.
  Assign
    qad_domain = global_domain
    qad_key1 = "ship_mstr"
    qad_key2 = shipnbr.
end.

update
qad_datefld[1]
qad_charfld[1]
with frame shipper.

See Also
STD-0305 Extending frozen schema tables to provide extra QAD fields (eB2 and up)
STD-0308 Proper Use of qad_wkfl (eB2 and up)
STD-0310 Progress source code file names in MFG-PRO

Summary

This standard contains guidelines for naming Progress source code files in MFG/PRO.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0310</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Environment, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Follow the guidelines below when naming Progress source code files in MFG/PRO.

- The format of a Progress source code file name should be 8.3 characters maximum, where the first 8 characters are a unique name, and the 3-character suffix represents what type of file it is. The 8.3 format is necessary because several MFG/PRO schema fields store program names and their format only allows 12 characters.

- The file name should be in lower-case only.

- The first 2 characters of all MFG/PRO program names represent the module that contains the program, for example "so" for Sales Orders and "ap" for Accounts Payable. The only exception to this rule is schema trigger programs, whose naming convention can be found in the referenced standard on "Progress Schema Triggers" below.

- The next 2 characters of menu-level programs generally identify the primary business entity being processed in the menu function. For example, soivpst.p is Invoice Post and apvomt.p is AP Voucher Maintenance.

- Current suffixes in use are .p for programs, .i for include files, .t for triggers, .v for schema validations & .w for GUI windows programs. However, a new suffix may be used as long as it is warranted and has been approved in the project's design.

- In general, certain menu-level programs of the following types should end in "*mt.p" for Maintenance, "*iq.p" for Inquiry, "*rp.p" for Report and "*up.p" for Batch Update (such as Delete/Archive). The inquiry and report menu-level programs may be appended with numerals to make them unique (e.g. sosorp01.p, sosorp02.p, etc.).

- If a new module is required, the designer must create a HelpDesk ticket for Development Systems so that the MFG/PRO product structure can be updated. The product structure database associates MFG/PRO menus with certain modules. This is not a trivial task and creating new modules should only be done with the proper authorization of the Architecture Team.

- Several types of programs in MFG/PRO have additional guidelines with regard to their naming. These include schema triggers, utilities, conversions, browses and certain business logic programs. These additional guidelines will appear in separate standards documents specifically related to the particular type of program, and are referenced below for more information.

- Proper naming conventions help to readily associate source code files with their module and their use. The module prefix on every source code file also helps to define MFG/PRO's modules and controls their deployment in many cases.

Examples

Right

Code
## Name | Function
--- | ---
sosomt.p | Sales Order Maintenance
sosoiq.p | Sales Order Inquiry
soivpst.p | Invoice Post
soivup.p | Invoice History Delete/Archive

### Wrong Code

<table>
<thead>
<tr>
<th>Name</th>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOMaint.p</td>
<td>mixed case, maintenance should be &quot;mt.p&quot;</td>
</tr>
<tr>
<td>SalesOrder.inng</td>
<td>not 8.3 format, mixed case, incorrect suffix</td>
</tr>
<tr>
<td>invoicePost.p</td>
<td>not 8.3 format, mixed case, wrong module prefix</td>
</tr>
<tr>
<td>ivivpst.p</td>
<td>no &quot;iv&quot; module</td>
</tr>
</tbody>
</table>

### See Also

STD-0032 Progress Schema Triggers
STD-0311 Progress selection-list widget and the Desktop

Summary

The Progress selection-list widget causes problems with the Desktop HTML engine and requires special handling.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0311</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Desktop UI Apps</td>
</tr>
<tr>
<td>Applicability</td>
<td>MFG/PRO versions eB and up</td>
</tr>
</tbody>
</table>

Description

When an MFG/PRO program uses a Progress selection-list widget, that widget will not be displayed in the Desktop HTML version of the program. This is due to the current limitations of the Desktop engine. There is a non-optimum but workable alternative that will function in the Desktop's HTML. The alternative is to use a scrolling window include file. This construct provides similar functionality to the Progress selection-list widget and is recognized and handled by the Desktop engine. An example of this is given below. This can be used instead of the Progress selection-list widget or can be used conditionally to run only when the program is running in Desktop (this is how the example uses it).

Note: This is not necessarily the only or best solution to this issue and if a more elegant Progress construct can be developed it will be included in this standard.

Rationale

The Desktop has a Progress engine that walks the widget tree of an MFG/PRO screen collecting information about the widgets so that it can render them in HTML. The engine does not handle selection-list widgets. So, if a selection-list widget is used in a frame in MFG/PRO it will not appear in the Desktop 2 HTML screen. The example is taken from program fstzsel.p in eB2. This code uses a scrolling window include file to implement a "selection-list" when Desktop is running (if(gpiswrap.i) = true) and uses a Progress selection-list for standard CHUI code. Only the relevant code is included here.

Examples

Right

Code

```plaintext
define variable tzsel as character view-as selection-list inner-chars 3 inner-lines 6.
form
  tzsel      at 2    no-label
  timezone   colon 14
with frame tzpopup side-labels width 43 overlay centered row pop-row
attr-space
title color normal (getFrameTitle("DISPLAY_TIME_ZONE",25)).
.
.
.
/* For html screens use an MFG/PRO selection list
```
* instead of a Progress selection list widget.
+ gpiswrap.i returns true if running in Desktop mode
+
if (gpiswrap.i) then do:
  hide frame tzpopup.
  run alternateSelectList(input tzsel:list-items,
    output timeZoneSelection).
  if timeZoneSelection = ? then leave.
  tzsel:screen-value = timeZoneSelection.
end.
else
  update
    tzsel
  with frame tzpopup.

PROCEDURE alternateSelectList:
/* -----------------------------------------------------------
Purpose      : Run the time zone selection list
Parameters   : pListValue - comma delimit list of
               values for the selection list
               pSelectedValue - (output) will be assigned
               the screen value chosen
               from the list
Notes        : pSelectedValue will be null (?) if the user
               F4's out of the selection list
-------------------------------------------------------------*/

define input parameter pListValues as character no-undo.
define output parameter pSelectedValue as character no-undo.

define variable j as integer no-undo.

for each tt-selectList exclusive-lock:
  delete tt-selectList.
end.

do j = 1 to num-entries(pListValues):
  create tt-selectList.
  selectionField = entry(j,pListValues).
end.

window_down = 6.

{ecwindow.i
  tt-selectList
  
  "selectionField format 'x(35)'"
  selectionField
  
  
  "yes"
  getFrameTitle("**DISPLAY_TIME_ZONE**",25)
  "no-labels width 40 centered attr-space"
  "timeZoneFrame"
  selectionField}

hide frame timeZoneFrame no-pause.

if keyfunction(lastkey) = "END-ERROR" then do:
  pSelectedValue = ?.
  return.
end.

find tt-selectList where recid(tt-selectList) = recidarray[i]
  no-lock no-error.
if available tt-selectList then
  pSelectedValue = selectionField.
END PROCEDURE.

Wrong

Code

/* IT IS WRONG TO USE ONLY A PROGRESS SELECTION-LIST WIDGET BECAUSE THE
PROGRAM WILL NOT
* WORK IN DESKTOP*/

define variable tzsel as character view-as selection-list inner-chars 3
inner-lines 6.

form
  tzsel at 2 no-label
  timezone colon 14
with frame tzpopup side-labels width 43 overlay centered row pop-row
attr-space
title color normal (getFrameTitle("DISPLAY_TIME_ZONE",25)).

update
  tzsel
with frame tzpopup.

See Also

STD-0290 Do not use the Progress Browse widget in maintenance program frames
STD-0312 Source file formatting - Header on files in RCS (eB2 and up)

Summary

The header of all source code files controlled by RCS, regardless of language, must conform to the format shown here.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0312</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress,C/C++</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Process,Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities in MFG/PRO versions eB2 and up</td>
</tr>
</tbody>
</table>

Description

The header of every source code file controlled by RCS, regardless of language, must be formatted exactly as shown below. The following notation conventions have been used:

- angle brackets < > indicate a placeholder for a mandatory item, which is to be replaced with the actual value (brackets should be removed)
- square brackets [] indicate a placeholder for an optional item, which is to be replaced with the actual value (brackets should be removed)
- an ellipsis (...) within a line indicates that the preceding item may be repeated as many times as needed
- an ellipsis on a line alone indicates that the preceding line may be repeated as many times as needed

All items within the header must be formatted as comments, and are shown here using the Progress comment delimiters /* */. If the programming language in use does not support this type of comment delimiter, appropriate comment delimiters should be substituted.

```c
/* <Filename> - <Brief Description>
/* Copyright 1986-<Year> QAD Inc., Carpinteria, CA, USA.
/* All rights reserved worldwide. This is an unpublished work.
/*[QAD Program Token - optional placement]
/*...
/*
/*<Detailed Description>
/*
/*<Revision History>
/*
/*[QAD Program Token - preferred placement]
/*...
/*<Blank Line>
```

Each item shown above is described in detail below, in the order that it must appear in the header.

- Filename - The filename of the source code file, including language-specific extension.
QAD Development Standards

- Brief Description - A short description of the purpose of the source code file, no more than a few words.
  Use mixed case. Appears on the same line as the filename.

- Year - The current year in YYYY format, to be placed as shown in the official QAD copyright statement.
  This field will be updated (if not the current year) anytime a source file is checked in.

- Detailed Description - A long description of the purpose of the source file, possibly multiple lines in length.
  Should be mixed case as in standard English. The format is described below.

- Revision History - Records revisions made to the file, including date changed and author, with a separate
  entry for each ECO. The format is described below.

- QAD program tokens (when needed) - Specific strings that tell the ECO processing system and other
  utilities how to handle the file. The preferred placement of these tokens on new files is below all
  commented lines of the header and just before the first executable line of the program. An alternate
  location for QAD tokens is to leave them at the top of the program just under the copyright statement. (See
  standards on tokens in the reference section below for more details on the tokens.)

Detailed Description and Revision History Formats
The detailed description consists of one or more lines of the form:

/* <Detailed Description Line>
 */
...

Where the placeholder Detailed Description Line represents the line(s)
containing the long description of the purpose of the source file.

The revision history consists of one or more entries, completed by a
comment signifying the end of the section. Each entry is a single line
containing one or more name-value pairs joined by colons, and separated by
one or more spaces. So, the revision history is formatted as follows:

/* <Name>:<Value> ...
 */
...
/*-Revision
 end---------------------------------------------*/

Where the placeholders shown are:

- Name - The name of an approved standard QAD revision history tag.
- Value - A value appropriate to the preceding tag.

Standard revision history tags are shown in the table below. In general, tags are required to appear only when
needed, and for several tags, the column position of and the order in which they appear is not mandated. There are
exceptions to this however, and certain tags are required to appear on all revision lines, or are required to be in
certain ordinal positions (not columns) on the line. The details are indicated in the table below.

Although it is not required, when modifying a program, try to align tags in the same ordinal and column positions on
all revision lines of that header. This will maximize the readability of the header.

<table>
<thead>
<tr>
<th>Revision History Tag</th>
<th>Description</th>
<th>Value (Format)</th>
<th>Required (Y/N or conditions)</th>
<th>Ordinal Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revision</td>
<td>The version of the file as recorded by RCS</td>
<td>Number (#.#...)</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td>BY</td>
<td>The full name of the person or utility that made the revision</td>
<td>String</td>
<td>Yes</td>
<td>2</td>
</tr>
<tr>
<td>DATE</td>
<td>The date on which the revision was recorded by RCS</td>
<td>Date (mm/dd/yy)</td>
<td>Yes</td>
<td>3</td>
</tr>
<tr>
<td>ECO</td>
<td>The ECO number under which the revision was made</td>
<td>String (<em>XXXX</em>)</td>
<td>For any changes made under an ECO</td>
<td>any</td>
</tr>
</tbody>
</table>
TIME | The time of day that the revision was made | Time (hh:mm) | No/Optional (some tools insert time automatically) | any

What the ‘ckout’ tool will do to a source file header in RCS

When a file is checked out of RCS, the ckout script will first update the ending year of the copyright statement in the header to the current year, if necessary. The stand-alone */$Revision: *" tag at the top of the program will also be removed, if it still exists in the file.

The main work of the ckout script will be to insert the new “Revision end” line into every source code file, if one does not already exist. This is necessary to allow the ‘ckin’ tool to automatically update revision history on the file. The ckout script will make every attempt to locate the appropriate place in the source file header to insert the “Revision end” line. If the attempt fails, the script will insert the “Revision end” line as the first line in the file.

Lastly, the ckout script will remove all "$" signs around any Revision tags in the code.

What a developer needs to do to a source file header in RCS

If creating a new source file, the developer should add the proper heading to the file in the format above, but should not include any revision history line (the ckin script will add this). The developer should also include a “Revision end” line using the exact format specified in this standard.

When modifying existing files, the developer should insure that the header matches the format described above. It is the responsibility of the developer to insure that the “Revision end” line is in the correct place at the end of the revision history section. If the ckout script added the “Revision end” line as the first line in the program and the developer did not relocate it, the ckin script will fail. In general, however, there should be no reason to modify or add revision history lines after the “Revision end” line is properly inserted, unless incorrect data was entered at the time of a previous ckin or the columns are misaligned. If the program does not contain a description, one should be added by the developer.

If a program needs to be obsoleted from a release, the developer simply needs to ckout and ckin the file without making any changes (the ckin script will add a revision history line). However, a comment indicating that the file is obsolete may be added above the first executable line of code.

When the ckin script is used to check in the file into RCS, the user can include an optional parameter to specify the full name of the developer, if the person executing ckin is not the same developer who made the change.

What the ‘ckin’ tool will do to a source file header in RCS

If no “Revision end” line exists in the file, the ckin script will fail.

If no ECO is provided with the ckin (as is the case in a VAT environment), then no further action will be performed to modify the file, and ckin will proceed as normal.

If an ECO is provided, a revision history line will be generated automatically and inserted into each source file by the ckin script during check-in into RCS. If the same ECO is used to make multiple changes to a source file, the ckin script will overwrite the last revision line with the updated data from the last check-in. This also applies when changes are merged from a VAT environment to the development baseline, if the same ECO is used in both releases. The revision history line from the VAT will be overwritten with a new line for the development baseline.

The ckin script will accept an ECO number and the full name of the developer, in addition to applying the correct date of check-in. If a developer name is not provided, the ckin script will use the user ID of the person performing the check-in to determine the developer’s full name. The ckin script will add "$" signs around the latest Revision tag so that RCS can update it with the new version number.

An example of this process is shown below.

The establishment of this uniform format for file headers makes it easy to determine the purpose and history of revisions for any given source file, even if you have not worked with it before. In addition, conformance to this standard will make it possible for tools to parse and extract the information contained in the header for a variety of purposes. The standard copyright statement provides legal protection for the intellectual property of the company. An example of a complete source file header is shown below for the Progress 4GL language. This is followed by an example of the life cycle of the “Revision” tag when used in the revision history.

Progress 4GL
Use of the "Revision" Tag

Examples of the revision history for source files when using RCS source code control are shown here at each stage in the source control process.

Header of newly created file as written by developer (Note: do not manually add a revision history line):

```
/* sosomtxx.p - Sales Order Custom Maintenance */
/* Copyright 1986-2003 QAD Inc., Carpinteria, CA, USA. */
/* All rights reserved worldwide. This is an unpublished work. */
/* */
/* */
/* This subroutine adds custom maintenance capabilities to MFG/PRO’s */
/* standard sales order maintenance functionality. */
/* */
/* */
/* Revision:1.0 BY:Sol Padune DATE:08/02/96 ECO:*XXXX* */
/* Revision:1.2 BY:Joe Developer DATE:11/05/00 ECO:*YYYY* */
/* $Revision:1.3$ BY:Mary Coder DATE:06/01/01 ECO:*ZZZZ* */
/*--Revision end---------------------------------------------------------------*/
/* */
/* */
/*V8:ConvertMode=Maintenance */
/* */
/*V8:RunMode=Windows */
```

Revision history after check-in for ECO XXXX:
Revision history after check-out, but before check-in of ECO YYYY (Note: do not manually add a new revision history line):

/* sosomtxx.p - Sales Order Custom Maintenance */
/* Copyright 1986-2003 QAD Inc., Carpinteria, CA, USA. */
/* All rights reserved worldwide. This is an unpublished work. */
/*
/* This subroutine adds custom maintenance capabilities to MFG/PRO's */
/* standard sales order maintenance functionality. */
/*
/* Revision:1.0 $ Revision:1.0$ BY:Sol Padune DATE:08/02/96 ECO:
*XXXX* */
/*-Revision
end-----------------------------------------------------------------*/
/*
/*
/*V8:ConvertMode=Maintenance */
/*V8:RunMode=Windows */
/*V8:RunMode=Windows*/

Revision history after check-in for ECO YYYY:
Revision history after second check-out and check-in for ECO YYYY (Note: breaks in the revision number sequence is acceptable in the header):

/* sosomtxx.p - Sales Order Custom Maintenance */
/* Copyright 1986-2003 QAD Inc., Carpinteria, CA, USA. */
/* All rights reserved worldwide. This is an unpublished work. */
/*
/* This subroutine adds custom maintenance capabilities to MFG/PRO's */
/* standard sales order maintenance functionality. */
/*
/* Revision:1.0  BY:Sol Padune  DATE:08/02/96  ECO: *XXXX* */
/* $Revision:1.1$  BY:Joe Developer  DATE:10/28/00  ECO: *YYYY* */
/*-Revision end---------------------------------------------*/
/*
*/
/*V8:ConvertMode=Maintenance */
/*V8:RunMode=Windows */

Revision history after check-out and check-in for ECO ZZZZ:
See Also

STD-0235 Proper usage of User Interface Tokens (RunMode)
STD-0236 Proper usage of GUI Converter tokens (ConvertMode)
STD-0196 Source file formatting - Header (pre-eB2)
STD-0316 Source file formatting - Header on files in Continuous CM (eB2 and up)
STD-0304 All CM-controlled files should have copyright & versioning
STD-0313 Double-Byte Languages Issues with FORMAT, STRING & INDEX

Summary

Developers must carefully consider the effect of Progress constructs FORMAT, STRING and INDEX when the number of characters shown on the user interface can differ from the number of bytes used to store the data, as in the case of double-byte languages.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0313</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Data Handling Logic, DB Fields, Double-Byte Languages, Field Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Progress supports double-byte languages in all of its 4GL constructs, but designers and developers need to be aware of how each construct acts upon the data stored in character fields and variables. In general, a single double-byte character will take up 2 bytes of storage and 2 columns of screen space in a character field or variable. The 2 bytes are typically referred to as the lead byte and trail byte. A double-byte character is invalid without both its lead and trail bytes.

**FORMAT**

The expression used to describe the format in the FORMAT construct represents the number of bytes to be displayed and entered when used with character fields and variables. For example, if a character schema field is defined with a format of "x(8)", the user will be allowed to enter 8 bytes of data through a user interface that makes use of this display format. This means that only 4 double-byte characters will be allowed through the user interface. Should we therefore restrict the use of formats like "x(1)" and "x(3)", since they will not appropriately allow for double-byte data? The answer is no, because in most cases fields with these formats are used to enter codes, such as status codes or currency codes. Typically, codes are used to provide a short-hand entry for the actual data they represent. This works fine for single-byte characters, but the process to enter double-byte characters via a keyboard does not allow for short-hand entry. In many cases, entry of a double-byte character requires 4 or 5 keystrokes through an IME (Input Method Editor). Therefore, most customers using double-byte languages will still rely on single-byte characters when entering these types of short-hand codes. Double-byte characters are typically reserved for fields that will be displayed on formal business documents, such as names, addresses and descriptions on orders and invoices.

**STRING**

The expression used to describe the format in the STRING construct represents the number of bytes to be retrieved when used with character fields and variables. For example, STRING(name,"x(1)") will return the first byte of the field 'name'. If the first character of name is a double-byte character, the function will return only the lead byte of the character, leaving the result invalid. If you want to retrieve the first 'n' characters of a string, consider using SUBSTRING instead of STRING, since the former allows retrieving by characters in addition to bytes (see the standard on LENGTH, SUBSTRING & OVERLAY in the reference section below).

**INDEX**

The INDEX function will return the number of character positions into a string where a search expression is located. For example, INDEX("ABC","B") will return 2, regardless of whether the characters A, B and C are single-byte or double-byte.

For more information on double-byte enabled applications, please see the Progress Internationalization Guide.

Developers must carefully consider the effect of Progress constructs on data that potentially might store double-byte characters. In some cases, misuse of the constructs may cause data corruption.

See Also

STD-0314 Double-Byte Languages Issues with LENGTH, SUBSTRING & OVERLAY
STD-0314 Double-Byte Languages Issues with LENGTH, SUBSTRING & OVERLAY

Summary

Developers must carefully consider the options presented with Progress constructs LENGTH, SUBSTRING and OVERLAY when the number of characters shown on the user interface can differ from the number of bytes used to store the data, as in the case of double-byte languages.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0314</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Data Handling Logic, Double-Byte Languages, Field Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Progress supports double-byte languages in all of its 4GL constructs, but designers and developers need to be aware of how each construct acts upon the data stored in character fields and variables. In general, a single double-byte character will take up 2 bytes of storage and 2 columns of screen space in a character field or variable. The 2 bytes are typically referred to as the lead byte and trail byte. A double-byte character is invalid without both its lead and trail bytes. The following Progress constructs allow the developer to specify whether they will use bytes or character units. It is important for the designer and developer to decide what they will work with when using these functions prior to the implementation. It is generally not acceptable to always use the default character units or rely on always using the RAW option to return bytes. Each circumstance might be different and requires careful planning for double-byte data.

**LENGTH**

The LENGTH function is able to return the number of characters units or bytes when used with character fields and variables. The CHARACTER parameter type to the function will return the number of character units in the expression (this is the default if no type is provided). The RAW parameter type to the function will determine the number of bytes in the expression. For example, LENGTH("ABC") will return a value of 3 character units, regardless of whether the characters A, B and C are single-byte or double-byte. LENGTH("ABC", "raw") will return 3 bytes if the characters are single-byte, and 6 bytes if they are double-byte. To determine the user interface size of labels as another example, typically the developer would use the RAW option, since screen columns equate to bytes. If the RAW option is not used, the length of a double-byte label may seem shorter since Progress will only count the number of characters. Double-byte characters typically occupy 2 columns on the screen.

**SUBSTRING & OVERLAY**

The SUBSTRING & OVERLAY constructs are very similar and are both able to work upon the number of characters units or bytes when used with character fields and variables. The CHARACTER parameter type to these functions will determine the number of character units in the expression to position the start and length of the affected string (this is the default if no type is provided). The RAW parameter type to the function will determine the number of bytes in the expression to position the start and length of the affected string. For example, SUBSTRING("ABC", 2, 1) = "X" will replace the second character "B" with "X", regardless of whether the characters A, B, C and X are single-byte or double-byte. SUBSTRING("ABC", 2, 1, "raw") = "X" will replace "B" with "X" just fine if the characters are single-byte. But if the characters are double-byte, "X" will actually overwrite the trail byte of "A" leaving the first character invalid. To illustrate some other uses of the type parameter, if the developer wanted the first character of a string, he would use the CHARACTER option or leave off the parameter to use this default. However, if the developer is storing fixed-byte lengths of strings by concatenating them into a single field or variable (something that should be avoided in new code), then he would use the RAW option to retrieve or replace fixed-byte sections of the data.

For information on the use of other parameter types, such as COLUMN and FIXED, please refer to the Progress Language Reference section on each of the constructs above. Additional Progress functions that operate on lists of strings, such as ENTRY or LOOKUP, do not need special consideration since these make use of delimiter strings to locate sections of data. Both the data and the delimiters can support double-byte characters without affecting the result of the function. For more information on double-byte enabled applications, please see the Progress Internationalization Guide.
**Rationale**

Developers must carefully consider the effect of Progress constructs on data that potentially might store double-byte characters. In some cases, misuse of the constructs may cause data corruption.

**Examples**

The MFG/PRO GL Report Writer is one module that has been found to require more attention to these types of constructs. This is because it uses text data entered by users to generate the reports. The example below comes from a correction that was implemented in this module. The variable scan-buffer contains the user-entered text, and we need to take into consideration the RAW parameter, otherwise the report will display incomplete data and will not be aligned properly.

**Right**

```plaintext
assign
state = "CENTERED"
k = truncate ((text_width - length(scan-buffer, "RAW")) / 2, 0)
text_out = text_out + fill("", k - length(text_out, "RAW")) + scan-buffer.
```

**Wrong**

```plaintext
assign
state = "CENTERED"
k = truncate ((text_width - length(scan-buffer)) / 2, 0)
text_out = text_out + fill("", k - length(text_out)) + scan-buffer.
```

**See Also**

STD-0313 Double-Byte Languages Issues with FORMAT, STRING & INDEX
STD-0316 Source file formatting - Header on files in Continuus CM (eB2 and up)

Summary

The header of all source code files controlled by Continuus CM, regardless of language, must conform to the format shown here.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0316</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>All</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Process, Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities in MFG/PRO versions eB2 and up</td>
</tr>
</tbody>
</table>

Description

The header of every source code file controlled by Continuus CM, regardless of language, must be formatted exactly as shown below. The following notation conventions have been used:

- angle brackets <> indicate a placeholder for a mandatory item, which is to be replaced with the actual value (brackets should be removed)
- square brackets [] indicate a placeholder for an optional item, which is to be replaced with the actual value (brackets should be removed)
- an ellipsis ... within a line indicates that the preceding item may be repeated as many times as needed
- an ellipsis on a line alone indicates that the preceding line may be repeated as many times as needed

All items within the header must be formatted as comments, and are shown here using the Progress comment delimiters /* */.

If the programming language in use does not support this type of comment delimiter, appropriate comment delimiters should be substituted.

```c
/* <Filename> - <Brief Description>

/* Copyright 1986-<Year>; QAD Inc., Carpinteria, CA, USA.

/* All rights reserved worldwide. This is an unpublished work.

/* %version: % for new files. Since
```

Each item shown above is described in detail below, in the order that it must appear in the header.

- Filename - The filename of the source code file, including language-specific extension.
- Brief Description - A short description of the purpose of the source code file, no more than a few words. Use mixed case. Appears on the same line as the filename.
- Year - The current year in YYYY format, to be placed as shown in the official QAD copyright statement. This field must be updated anytime a source file is changed.
- Version Tag - The current version of the file as defined by the Continuus CM tool. The tag must be placed inside of percent signs, and with a space following the colon, e.g. %version: % for new files. Since
Continuus is case sensitive, this tag must always be all lower case. Upon check-out, Continuus will update this tag with the current version number for the file, e.g. %version: 1.1%.

- Detailed Description - A long description of the purpose of the source file, possibly multiple lines in length. Should be mixed case as in standard English. See details below for differences in source code files.
- Author Name - The full name of the person who created this revision of the file.

**Detailed Description on Java Source Files**

```java
/**
 * <Detailed Description Line>
 * ...
 * @author <Author Name>
 */
```

- Detailed Description Line(s) - The detailed description for Java source code files should follow the format required by the JavaDoc documentation generation utility from Sun Microsystems. See the Sun website for more details on JavaDoc (http://java.sun.com/products/jdk/1.2/docs/tooldocs/solaris/javadoc.html).
- Author Name - The full name of the person who created this revision of the file.

**Detailed Description on Non-Java Source Files**

```c
/* <Detailed Description Line> */
...```

- Detailed Description Line(s) - Description of the purpose of the source file.

⚠️ Detailed revision history of the source file will be maintained only in the Continuus CM tool.

**Rationale**

The establishment of this uniform format for file headers makes it easy to determine the purpose and current version for any given source file, even if you have not worked with it before. In addition, conformance to this standard will make it possible for tools to parse and extract the information contained in the header for a variety of purposes. The standard copyright statement provides legal protection for the intellectual property of the company. Examples of complete source file headers are shown below for the Java and C languages.

**Examples**

**Java**

```java
/* program.java - Widget Manager */
/* Copyright 1986-2002 QAD Inc., Carpinteria, CA, USA. */
/* All rights reserved worldwide. This is an unpublished work. */
/* */
/* */
/* %version: 1.2% */
/**
 * This class is responsible for tracking and managing all widgets
 * in the user interface. Acts as a dictionary of widgets.
 * @author A.N. Arthur
 */
```

**C**

Note that unlike the previous example, no percent signs appear in the revision history. The developer has copied the latest version number (2.0) into the revision history by hand.
See Also

STD-0196 Source file formatting - Header (pre-eB2)
STD-0312 Source file formatting - Header on files in RCS (eB2 and up)
STD-0304 All CM-controlled files should have copyright & versioning
STD-0320 Domain Should Be Considered In All Queries (eB2.1 and up)

Summary

The release of MFG/PRO eB2.1 introduces a new data object, Domain, into almost all MFG/PRO tables & indexes. This should be taken into account when writing procedures to access data, in order to avoid performance problems.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0320</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>3</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Data Handling Logic, DB Indexes, MFG/PRO Design, Table Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities for MFG/PRO versions eB2.1 and up.</td>
</tr>
</tbody>
</table>

Description

The release of MFG/PRO eB2.1 introduces a new data object, Domain, into MFG/PRO (table dom_mstr). Most of the other data objects in MFG/PRO are linked to the Domain object through a “belongs to this domain” relationship.

A field to hold the domain value has been added to the majority of the tables in the MFG/PRO schema. The field name used in each case follows the QAD naming standards (e.g. cm_domain, pt_domain, etc.).

These new fields have become the first component in all the indexes on the tables to which they have been added. To access the data in such tables, and to use these indexes efficiently, the domain field must form a part of any data query.

A main OID field and corresponding index were added to every qaddb table in eB3 and the Domain field is not a component of this main OID index. By design, the OID field value is unique across the entire database and is intended as a single unique record identifier.

The Domain object has been introduced in order to partition information in the MFG/PRO database by Domain. This object can be used to represent different legal businesses within a single MFG/PRO database. In previous versions, such representations require the use of separate MFG/PRO databases.

This concept works by introducing the new data architecture and a new globally scoped variable; global_domain. This variable is set to one of the Domain values at the point which a user logs in to MFG/PRO. From that point on, all data access statements should limit the data retrieved to the global_domain value (or a domain value suitable for the application involved).

Solution:

For tables with a domain field:

1. Always structure a WHERE or USING clause to have the domain field as the first conditional in the query.
2. Do not allow any record selection statements on tables with a domain field to omit domain from the selection parameters, unless required for the application.
3. Always perform an equality check against the global_domain variable, unless some other domain variable is more suitable (for the application).
4. Always make sure the domain conditional has logical priority over the other conditionals in the query.

Exceptions:

Tables that contain a main OID field and corresponding index can be accessed without specifying the Domain value as long as this main OID index will be used in the retrieval. The main OID index will never contain a Domain field and in fact, can only contain the main OID field. For example, when an OID value is known, a retrieval of that record can just specify this OID field value in the where clause.

When two tables are related by a foreign key OID field, if the retrieval of related records will use the OID indexes, then the Domain field does not need to be specified in the where clause. An example would be a parent-child relationship where the OID field of the parent table is included as a foreign key OID field in the child table along with the required indexes. If a parent record is already in scope, the retrieval of the related child records can use
the Progress 'OF' construct as in 'for each <child_table> OF <parent_table> no-lock:' without specifying a where clause. This retrieval will use the foreign key OID index in the child table that does not contain a Domain field. For more information see DST-0005 Developing with OID fields (eB3 and up).

The domain fields can be omitted if the application requires it (i.e. the intention is to retrieve data from all domains, etc.). However, in that case careful consideration should be given to what index will be used if domain is omitted, to ensure good performance.

Certain tables do not have a domain field (because they apply over the whole system). A list of tables which have the domain field, and those that do not, can be found in the Domain design document: Multiple Domains (SO123) Design Specification. Note that this document is current only during the initial release of eB2.1.

**Rationale**

To implement partitioning of data in MFG/PRO by domain, but to preserve efficient use of queries and data retrieval.

**Examples**

**Right**

**Code**

Example 1 – domain selection must appear first in the where clause:

```
for each pt_mstr
  fields (pt_domain pt_prod_line pt_part pt_part_type pt_group)
  no-lock
  where pt_domain = global_domain
  and (pt_prod_line &gt;= line and pt_prod_line &lt;= linel)
  and (pt_part &gt;= part and pt_part &lt;= part1)
  and (pt_part_type &gt;= type and pt_part_type &lt;= typel)
  and (pt_group &gt;= ptgroup and pt_group &lt;= group1)
  break by pt_prod_line by pt_part:
    *
    *
    *
end.
```

Example 2 – match against global_domain or another valid domain. In the following example, this eCommerce application uses a different variable to hold a domain value (set at some point in the program). This is because the domain this routine should process is NOT the current domain, but a different target domain. By using a separate variable, the current domain value is preserved (in global_domain), but the correct data can be processed. However, the target must still be a valid domain value.

```
for each ptp_det
  fields (ptp_domain ptp_part ptp_site ptp_network ptp_pm_code)
  no-lock
  where ptp_domain = ecom_domain
  and ptp_part = part
  and ptp_site &gt;= site
  and ptp_network &gt;= ""
  and ptp_pm_code = "D":
    *
    *
    *
end.
```

Example 3 – domain conditional must have logical priority (i.e. must apply to the rest of the query no matter what "OR" conditions exist). If in any doubt, enclose the rest of the conditions with brackets.
for each tr_hist
fields (tr_domain tr_nbr tr_line tr_type tr_addr)
exclusive-lock
where tr_domain = global domain
AND ( tr_type = "ORD-SO" or tr_type = "ISS-SO" )
break by tr_nbr:

Wrong

Code

Example 1 – domain does not appear in the where clause (causes whole-index search):

for each pt_mstr
fields (pt_domain pt_prod_line pt_part pt_part_type pt_group)
no-lock
where (pt_prod_line &gt;= line and pt_prod_line &lt;= line1)
and (pt_part &gt;= part and pt_part &lt;= part1)
and (pt_part_type &gt;= type and pt_part_type &lt;= type1)
and (pt_group &gt;= ptgroup and pt_group &lt;= group1)
break-by pt_prod_line by pt_part:

Example 2 – domain not matched against global_domain or another valid domain value (causes **ptp_det not available):

for each ptp_det
fields (ptp_domain ptp_part ptp_site ptp_network ptp_pm_code)
no-lock
where ptp_domain = "some invalid constant etc."
and ptp_part = part
and ptp_site &lt;=&gt; site
and ptp_network &gt; ""
and ptp_pm_code = "D":

Example 3 – domain conditional does not have logical priority (causes records from other domains to be selected as well):

for each tr_hist
fields (tr_domain tr_nbr tr_line tr_type tr_addr)
exclusive-lock
where tr_domain = global domain
and tr_type = "ORD-SO" or tr_type = "ISS-SO"
break-by tr_nbr:

See Also

DST-0006 Design Considerations for Domained MFG-PRO Versions (eB2.1 and up)
STD-0323 Switching domains in procedure code
DST-0005 Developing with OID fields (eB3 and up)
STD-0322 Temp-table names and their field names should be 50 characters or less

Summary

Progress temp-tables should be defined with names that are 50 characters or less. Temp-table fields should also be defined with names that are 50 characters or less.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0322</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Tables</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

When defining temp-tables in Progress code, ensure that the name of the temp-table is 50 characters or less. Each field in the temp-table also needs to be named with 50 characters or less. When versions of MFG/PRO are released, the Progress XREF data from the system compile is used to populate data in several qadhelp database tables, which will then be used to support the System Cross-Reference functions within MFG/PRO. Temp-tables and their field data is included in the XREF data obtained from the compile. If their names are larger than 50 characters, those data records will cause loading errors when populating an Oracle qadhelp database since the SQL-WIDTH for the qadhelp tables involved only support 50 characters.
STD-0323 Switching domains in procedure code

Summary

When there is a need to switch the current domain to another domain (and back again) in a multiple domain environment, use the standard procedure gpmdas.p.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0323</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Language</td>
<td>Progress,Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>MFG/PRO Application,Procedure Structure</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities for eB2.1 and above.</td>
</tr>
</tbody>
</table>

Description

MFG/PRO eB2.1 introduced the “domain” data object as a new way of partitioning data. From eB2.1 onwards almost all data in MFG/PRO (sites, entities, accounts, inventory etc) can split by domain. Prior to eB2.1, a site would be associated with a database in Site Maintenance. From eB2.1 onwards, a site is associated with a domain, and the domain is associated with a database in Domain Maintenance, using the dom_db field.

A dom_mstr record for a domain in the database the domain is associated with is called a “Primary” record (i.e. dom_db matches the database it is in). A dom_mstr record for a domain in a different database to that associated with the domain is called “Secondary” (i.e dom_db does not match the database it is in). Any transaction data that is site dependent needs to be accessed (read, create, update, delete) in the domain where that site belongs. The domain which owns the site is called the primary domain of a site. The primary domain is currently stored on the site record in the si_db field. If the site domain field (si_db) is different from your current domain (as stored in the global variable global_domain), then it may be necessary to switch the current domain to the site domain.

There is a single procedure that will switch the current domain to a target domain. This procedure is called “gpmdas.p”. The MDAS acronym stands for Multiple Domain Alias Switching. The purpose of the program is to switch the current domain to a target domain. The program will also, if necessary, switch the QADDB alias to the database that holds the target domain. This single procedure replaces the gpalias family of programs. In fact the 4 gpalias procedures still exist, but the procedures have been re-engineered to use the gpmdas.p procedure. The gpalias programs are now deprecated from eB2.1 onwards and should never be used in new programs.

When the gpmdas procedure is used, the current domain is switched to the target domain. Validation is done to ensure that the target domain does exist and that the domain is active. Once this validation has been done, a check is then done to see if the domain is primary in the current MFG/PRO database, or if the domain is primary in another database. Validating the domain database field (dom_db) in the domain master record (dom_mstr) for the target domain carries out this check. If the database of the target domain is not the current database, then the QADDB alias will be switched to the new database.
The gpmdas.p procedure has been designed to handle issues involved with switching the current domain as well as the switching of the QADDB alias. The principal issue with the switching of the domain is that other variables (such as variables with global scope) and records must be set when the domain is switched.

The gpmdas.p procedure will output one of the following error statuses:

Status 0: The domain was changed.

Status 2: The destination domain does not exist.

Status 3: The database for the destination domain is not available or not connected.

Status 9: This is the default status of the procedure.

Always remember to switch back to the original domain (via a second call to gpmdas.p) once all processing has been completed in the new domain. Failure to do this will result in the next program to be executed to be run in a different domain than the expected one.

Regarding the future of the global_db field: In versions prior to eB2.1, the global_db field was used to store the current working database that the user was using. In eB2.1, this has changed. The global_db field will store the current domain in which the user is working. In effect, the value of the global_db field will be equal to the value stored in the global_domain field. In future versions of MFG/PRO, the use of the global_db field will be rationalised; the variable global_domain will be storing the current working domain long term with the variable global_db being made obsolete.

All the complexity and issues of switching domains (and possibly the database as well) are encapsulated in a single procedure. If the switching logic has to change we only have to do it in this single procedure.

1. Make sure you have the originating domain in a field or variable that you don’t manipulate with the processing in other domains.
2. Get the site record for the site you want to process against.
3. Call gpmdas.p to switch the domain to the domain held in the si_db field if necessary.
4. Call the new program that will do the processing in the “other” domain.
5. Call gpmdas.p to switch the domain back to the originating domain, i.e. the domain saved in step 1.

Global_db is a global variable that always contains the domain of the current domain. gpmdas changes the value of global_db whenever you switch domains. The other global variable that is also used for this purpose is global_domain.

A) Do not explicitly re-assign the global_domain and global_db variables to the new domain value.

B) Do not explicitly use the CREATE ALIAS statement if the target domain exists in another database. This will create a problem under Oracle.

C) Do not try to read or update tables where the records are supposed to be found in the “other” domain or DB. This will look in the table of the original DB where it typically won't find any records or it will find wrong or obsolete records. You need to call a new program to be able to access tables in the new qaddb and domain.
D) Don't forget to switch back to the originating domain and database. If you don't switch back, the next program the user runs will be against a different domain and database than the user expects.

E) The gpalias programs should never be used in new programs and procedures to switch the domain and database.

```lisp
original_db = global_db.
for each si_mstr no-lock where ......:
    /* Switch to the Inventory site */
    if si_db <> global_db and si_db <> "" then do:
        find first dom_mstr where dom_domain = si_db
        no-lock no-error.
        If available dom_mstr then do:
            A) assign global_domain = si_db
            global_db = global_domain
            B) if dom_db <> sdbname("qaddb") then do:
                create alias QADDB for si_db.
            end.
            end.
            C) for each ld_det where ld_domain = global_domain
               and ld_site = si_site:
                total_qty = total_qty + ld_qty_oh.
            end.
            end.
    D) /* Now switching back to original Domain and DB */
    E) {gprun.i ""gpalias.p"
```

See Also

STD-0006 Allowed number of fields per table
STD-0320 Domain Should Be Considered In All Queries (eB2.1 and up)
STD-0145 Multiple DB - Switching aliases (pre-eB2.1)
STD-0325 Use OID fields as foreign keys to relate qaddb tables (eB3 and up)

Summary

Any table relationship introduced to qaddb schema starting in eB3 should use a foreign key field that points to the main OID field that uniquely identifies a record in the related table.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0325</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>3</td>
</tr>
<tr>
<td>Language</td>
<td>Progress,Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields, DB Indexes, DB Tables, MFG/PRO Application, MFG/PRO Design, Table Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities in eB3 and up</td>
</tr>
</tbody>
</table>

Description

Starting in eB3, every qaddb table will have a main OID field along with a unique index consisting of the single field. Any table relationship introduced to qaddb schema, using new or existing tables, should use a foreign key field that points to the main OID field in the related table. The field name for the foreign key field will usually have the exact same name as the main OID field in the related table. For example, if a new table will be linked to the Sales Order Detail table that, starting in eB3 contains an oid_sod_det field, the new table should also contain a field named oid_sod_det along with an index starting with this field. The oid_sod_det field in the new table is referred to as a foreign key OID field to the sod_det table. The attributes of the foreign key OID field should exactly match those of the main OID field in the related table except for the description which should follow the convention below.

Foreign Key OID Field conventions:

- Name: oid_<related_table>
- Description: "Foreign key reference to unique record identifier in <related_table>.
- Index Name: <prefix>_oid_<related_table>
- Index-Field: oid_<related_table>

Where:

- <related_table> is the Progress name of the related table
- <prefix> is the prefix for this table where the foreign key field is being added (For example: 'gl' for table 'gl_ctrl')

The main OID field in a related table is always named oid_<related_table> and so the foreign key OID field name will exactly match the main OID field in the related table. This means that multiple tables in MFG/PRO may now contain a field with the exact same name. But this will only be allowed for fields starting with "oid_". For this reason, all OID fields should be qualified with the table name when referenced in the code.

An index should be added for each foreign key OID field that is added to a table and must consist of the foreign key OID field optionally followed by additional fields. This index does not have to be unique. This will allow the Progress "OF" construct to be used in retrievals in place of explicitly listing the join condition. For example:

```sql
/* <table> record is already in scope */
for first <related_table> of <table> no-lock:
```

or

```sql
/* <table> record is already in scope */
for first <related_table> of <table> no-lock:
```
/* <related_table> record is already in scope */
for each <table> of <related_table> no-lock:

or

for each <related_table> where ... no-lock,
each <table> of <related_table> no-lock:

can be used as long as <related_table> contains a main OID field (e.g. oid_<related_table>) and unique index and
<table> contains a foreign key field (e.g. oid_<related_table>) and an index starting with the oid_<related_table>
field.

The name of the foreign key OID index should always match the name of the foreign key OID field.

Special Situations

1. Multiple foreign key OID fields pointing to the same table:
   If two or more foreign keys are needed to reference multiple records from the same table, one of the foreign keys
   should be named as described above, the other foreign keys to this same table will use this same name plus
   meaningful suffixes. For example, if we were to design so_mstr from scratch, we might need two foreign keys to the
   cm_mstr table: one for the sold-to customer and a second for the bill-to customer. These two foreign keys residing
   in so_mstr could be named: oid_cm_mstr and oid_cm_mstr_bill. We might also need a foreign key to the ad_mstr
   table for the ship-to address. This foreign key would be named oid_ad_mstr.

   Multiple Foreign Key OID Field conventions:

   Name: oid_<related_table>[<_suffix>]
   Description: "Foreign key reference to unique record identifier in
   &lt;related_table&gt;."
   Index Name: &lt;prefix&gt;oid_<related_table>[<_suffix>]
   Index-Field: oid_<related_table>[<_suffix>]
   [...]

   where

   &lt;related_table&gt; is the table name of the related table

   _suffix is only used if there are multiple foreign key OID fields for the same related table. One of these fields must
   be named without the _suffix.

   &lt;prefix&gt; is the prefix for this table where the foreign key field is being added (e.g. 'gl' for table 'gl_ctrl')

   For the relationship defined by the foreign key OID field with a suffix, the Progress "OF" construct cannot be used
   and the retrieval must explicitly specify the join condition as in:

   /* <table> record is already in scope */
   for first <related_table> where <related_table>.oid_<related_table> =
   <table>.oid_<related_table>[<_suffix>] no-lock:

2. Each record will be related to one of multiple tables:
   If a table is to be related to two (or more) tables but each record will only be related to one of the tables, then
   instead of adding a foreign key OID field for each of the possible relationships, a more generic foreign key OID field
   should be used along with a field identifying the related table. An example would be the mrp_det table where each
   record is related to one of a number of tables as shown in the ER diagrams in the File Relationships manual. If
   these relationships were defined by a generic foreign key OID pair, then exactly two fields would be needed to
   represent all of the relationships: oid_dataset to contain the OID value for the related record, and mrp_dataset to
   contain the table name for the related record. Currently, each of these relationships is defined by a minimum of 5
   fields.

   Generic Foreign Key OID Pair conventions:
Name: oid_dataset
Description: "Foreign key reference to unique record identifier in the table identified in <prefix>_dataset."

Name: <prefix>_dataset
Description: "Identifies the table containing the related record identified in oid_dataset."
Index Name: <prefix>_oid_dataset
Index-Field: oid_dataset

Where:

<prefix> is the prefix for this table where the foreign key field is being added (For example: 'gl' for table 'gl_ctrl').

In the unlikely event that more than one generic foreign key OID pair is needed in a table, then a suffix can be added to these names.

In this situation, the Progress "OF" construct cannot be used and the retrieval must explicitly specify the join condition as in:

```sql
if mrp_dataset = "sod_det" then
  for first sod_det where mrp_det.oid_dataset = sod_det.oid_sod_det
no-lock:
```

Or a dynamic query can be constructed to make use of the value of mrp_dataset in the actual query.

**Rationale**

Because each qaddb table now has a single unique OID field, table relationships no longer need to consist of multiple fields and should always use the OID fields.

By using OID fields to relate records, the values of key business fields can be changed in a master table without having to change corresponding foreign key values in all related records. The main OID value will not change.

By using the exact same OID field name in each of the related tables along with indexes on the OID fields, the Progress "OF" construct can be utilized to simplify joins. This also allows 3rd party relational database tools to detect these table relationships.

**Example**

**Right**

**Design**

Assume that a new contact table, adcd_det, is being added that relates to the ad_mstr table. The new contact table may contain 0 to n related records for every ad_mstr record. The ad_mstr table contains an oid_ad_mstr field along with a unique oid_ad_mstr index for the field. The new adcd_det table might contain the following entries with the compliance for this standard shown in **red**:

```sql
ADD TABLE "adcd_det"
AREA "Schema Area"
LABEL "Address Contact Detail"
DESCRIPTION "Address Contact Detail"
DUMP-NAME "adcd_det"
TABLE-TRIGGER "Create" NO-OVERRIDE PROCEDURE "adcdc.t" CRC "?"

ADD FIELD "oid_adcd_det" OF "adcd_det" AS decimal
DESCRIPTION "Unique application generated record identifier."
FORMAT "))))))))))))))))))))9.9``
INITIAL "0"
MAX-WIDTH 38
DECIMALS 10
MANDATORY

ADD FIELD "oid_ad_mstr" OF "adcd_det" AS decimal
DESCRIPTION "Foreign key reference to unique record identifier in ad_mstr."
```
FORMAT ">>>>>>>>>>>>>>>>>>>>>>>>>>>9.9<<<<<<<<<"
INITIAL "0"
MAX-WIDTH 38
DECIMALS 10
MANDATORY

ADD FIELD "adcd_contact_name" OF "adcd_det" AS character
DESCRIPTION "Name of contact."
FORMAT "x(35)"
INITIAL ""
LABEL "Contact Name"
MAX-WIDTH 35
MANDATORY

... ADD INDEX "oid_adcd_det" ON "adcd_det"
UNIQUE
INDEX-FIELD "oid_adcd_det" ASCENDING

ADD INDEX "adcd_oid_ad_mstr" ON "adcd_det"
PRIMARY
UNIQUE
INDEX-FIELD "oid_ad_mstr" ASCENDING
INDEX-FIELD "adcd_contact_name" ASCENDING

Code
/* To create a new related record */
for first ad_mstr where ad_addr = input ad_addr exclusive-lock:
create adcd_det.

assign
adcd_det.oid_ad_mstr = ad_mstr.oid_ad_mstr.
update adcd_det except adcd_det.oid_adcd_det.
...
end.

/* To retrieve a related record to ad_mstr */
for each adcd_det of ad_mstr no-lock:
display adcd_det except adcd_det.oid_adcd_det.
end.

/* To retrieve a set of ad_mstr records along with their related adcd_det records */
for each ad_mstr fields (ad_addr, ad_name) where ad_addr >= fromvalue and
ad_addr <= tovalue no-lock,
each adcd_det fields (adcd_contact_name) of ad_mstr no-lock:
display ad_addr ad_name ad_contact_name.
end.

Wrong

Design

The following new table definition defines a foreign key to the ad_addr field in ad_mstr instead of the oid_ad_mstr field. The violations of this standard are shown in red.

ADD TABLE "adcd_det"
AREA "Schema Area"
LABEL "Address Contact Detail"
DESCRIPTION "Address Contact Detail"
DUMP-NAME "adcd_det"
TABLE-TRIGGER "Create" NO-OVERRIDE PROCEDURE "adcdc.t" CRC "?"

ADD FIELD "oid_adcd_det" OF "adcd_det" AS decimal
DESCRIPTION "Unique application generated record identifier."
FORMAT ">>>>>>>>>>>>>>>>>>>>>>>>>>>>9.9<<<<<<<<<"
INITIAL "0"
MAX-WIDTH 38
DECIMALS 10
MANDATORY
ADD FIELD "adcd_addr" OF "adcd_det" AS decimal
DESCRIPTION "The unique code that identifies an address."
FORMAT "x(8)"
INITIAL ""
LABEL "Address"
MAX-WIDTH 80
MANDATORY

ADD FIELD "adcd_contact_name" OF "adcd_det" AS character
DESCRIPTION "Name of contact."
FORMAT "x(35)"
INITIAL ""
LABEL "Contact Name"
MAX-WIDTH 35
MANDATORY

... 

ADD INDEX "oid_adcd_det" ON "adcd_det"
UNIQUE
INDEX-FIELD "oid_adcd_det" ASCENDING

ADD INDEX "adcd_addr" ON "adcd_det"
PRIMARY
UNIQUE
INDEX-FIELD "adcd_addr" ASCENDING
INDEX-FIELD "adcd_contact_name" ASCENDING

Code

define ado_mstr where ad_addr = input ad_addr exclusive-lock:
    create adcd_det.
    assign adcd_addr = ad_addr.
    update adcd_det except adcd_addr.
...
end.

for each adcd_det where adcd_addr = ad_addr no-lock:
    display adcd_det except adcd_addr.
end.

See Also

DST-0005 Developing with OID fields (eB3 and up)
STD-0050 Required Table Fields
STD-0180 Every table must have a unique index & a primary index (pre-eB3)
STD-0326 Every qaddb table must have a create trigger (eB3 and up)

Summary

Every table in the qaddb database must have a create trigger that will populate the main OID field in new records. The create trigger must follow a strict coding template that uses generic include files.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0326</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>3</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Sequences, DB Tables, MFG/PRO Design, Table Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities in eB3 and up</td>
</tr>
</tbody>
</table>

Description

Every new table must contain a create trigger to populate the main OID field in new records. This create trigger must be defined in the schema and the corresponding .t file must use the generic include files provided in the eB3 baseline.

Schema Definition

The 'ADD TABLE' definition in the .df file must define a create trigger with the following format:

```
ADD TABLE "<table_name>"
...
  TABLE-TRIGGER "Create" NO-OVERRIDE PROCEDURE "<prefix>c.t" CRC "?"
where:
<table_name> is the Progress table name
<prefix> is the prefix for the table name (e.g. <prefix> = gl when 
<table_name> = gl_mstr)
```

Trigger Source File

The name of the create trigger source file must match the name provided in the schema definition ("<prefix>c.t"). The logic in the new source file should follow this coding template:

```
/* <prefix>c.t - CREATE TRIGGER FOR TABLE <table_name> */
...
/* This trigger code is executed when create event occurs for the table. */
.../
/*-Revision
end-------------------------------------------------------------*/
/*V8:ConvertMode=NoConvert*/

TRIGGER PROCEDURE FOR CREATE OF <table_name>.
{gpoidfcn.i} /* Contains nextOidValue function */
{gpoidcr.i &TABLE-NAME=<table_name>}
```

Where:

<table_name> is the Progress table name
<prefix> is the prefix for the table name (e.g. <prefix> = gl when <table_name> = gl_mstr)

Note: mfdeclre.i is not referenced in the create trigger. This is intentional so that the create trigger will execute without error if a record is created from a Progress Editor session.

**Rationale**

Create triggers allow OID fields to be introduced to legacy MFG/PRO in a very non-invasive way. The Generic OID population routine ensures consistent and unique values. By using a create trigger as opposed to a write trigger, the new field value is available in the code immediately after the create statement so that the value can be used to populate foreign key fields in related records.

**Examples**

Assume that a new contact table, adcd_det, is being added to the qaddb database in eB3.

**Schema:**

```
ADD TABLE "adcd_det"
AREA "Schema Area"
LABEL "Address Contact Detail"
DESCRIPTION "Address Contact Detail"
DUMP-NAME "adcd_det"
TABLE-TRIGGER "Create" NO-OVERRIDE PROCEDURE "adcdc.t" CRC "?"
```

```
ADD FIELD "oid_adcd_det" OF "adcd_det" AS decimal
DESCRIPTION "Unique application generated record identifier."
FORMAT ">>>>>>>>>>>>>>>>>>>>>>>>>>>>9.9<<<<<<<<<"
INITIAL "0"
MAX-WIDTH 38
DECIMALS 10
MANDATORY
...
```

```
ADD INDEX "oid_adcd_det" ON "adcd_det"
UNIQUE
INDEX-FIELD "oid_adcd_det" ASCENDING
```

**Code:**

Contents of new trigger source file adcdc.t:

```
/* adcdc.t - CREATE TRIGGER FOR TABLE adcd_det */
/* Copyright 1986-2004 QAD Inc., Carpinteria, CA, USA. */
/* All rights reserved worldwide. This is an unpublished work. */
/* $Revision: $ */
/* */
/* This trigger code is executed when create event occurs for the table. */
/* */
/* $Revision: $ BY: Sandy Brown (sbb) DATE: 09/16/04 ECO: SBBB */
/* Revision end */

/V8:ConvertMode=NoConvert/

TRIGGER PROCEDURE FOR CREATE OF adcd_det.

(gpoidfcn.i) /* Contains nextOidValue function */

(gpoidcr.i &TABLE-NAME=adcd_det)

**See Also**

DST-0005 Developing with OID fields (eB3 and up)
STD-0032 Progress Schema Triggers
STD-0050 Required Table Fields
STD-0327 Every table must have a unique index & a primary index (eB3 and up)
STD-0327 Every table must have a unique index & a primary index (eB3 and up)

Summary

Every table must have a unique index and a primary business index.

- For MFG/PRO qaddb tables, this main unique index must consist of only the main OID field for the table and the index name must match this OID field name.
- For QAD Financials tables, this main unique index must consist of only the <tablename>_ID field and the index name is typically "Prim".

The primary business index should be a separate index and does not need to be unique. However, it is strongly recommended that the primary business index be unique whenever possible.

Having a primary, unique index will aid database-specific auditing processes. For MFG/PRO non-qaddb tables, the main unique index should not contain the main OID field because that field will only be populated with the default value of zero. In these non-qaddb tables, the primary business index will often be the same as the unique index.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0327</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2</td>
</tr>
<tr>
<td>Language</td>
<td>Progress, Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields, DB Indexes, DB Tables, MFG/PRO Design, Table Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities starting in eB3</td>
</tr>
</tbody>
</table>

Description

**All QAD Product Index Requirements:**

Every table should have a primary index that contains fields reflecting the most common access of the table's records via business logic. In most cases, the main unique index will also be primary.

However, the primary index does not have to be unique, but the Database Team strongly recommends that it is unique.

**MFG/PRO Unique and Primary Index Requirements:**

Starting in eB3, every new table will have a main OID field defined in the schema. See STD-0050: Required Table Fields.

**Qaddb Tables:**

For qaddb tables, the main OID field will contain a unique system-generated record identifier. A unique index that consists of only this main OID field must be added to each new table to guarantee record uniqueness. This unique index is referred to as the main OID index. This does not preclude the addition of other unique indexes. The name of the main OID index must match the main OID field name exactly and will not begin with the table prefix.

Foreign key indexes that consist of foreign key OID fields will always begin with the table prefix. In this way it is easy to distinguish the main OID index in the table from foreign key indexes: the main OID index will start with "oid_" and foreign key OID indexes will start with the table prefix.

See STD-0325: Use OID fields as foreign keys to relate qaddb tables (eB3 and up).

There should be a single primary index on every table that contains fields reflecting the most common access of the table's records via business logic. In most cases, the main OID index should not be primary. The primary index does not have to be unique, but the Database Team strongly recommends that it is unique in order to aid database auditing processes.
A table’s primary index may not be suitable for a database’s native auditing feature, so the audit index may differ from the table’s primary one.

Non-qaddb Tables:

For non-qaddb tables, the main OID field will only be populated with the default value of zero and will not contain a unique value. For these tables, the unique index must not contain the main OID field and instead must consist of the minimum number of key fields to guarantee record uniqueness. Do not include fields that are convenient for sorting in this main unique index.

Every table should have a primary index that contains fields reflecting the most common access of the table’s records via business logic. In most cases, the main unique index will also be primary.

However, the primary index does not have to be unique, but the Database Team strongly recommends that it is unique.

**QAD Financials Unique and Primary Index Requirements:**

Every table requires a primary index named “Prim” containing only the “<table-name>_ID” field. For example:

```
ADD INDEX "Prim" ON "Profile"
AREA "FIN_IDX"
UNIQUE
PRIMARY
INDEX-FIELD "Profile_ID" ASCENDING
```

Provide a non-primary but unique index on all tables when possible. For example:

```
ADD INDEX "Code" ON "Profile"
AREA "FIN_IDX"
UNIQUE
INDEX-FIELD "ProfileCode" ASCENDING
```

**Rationale**

This standard ensures sure that no program can accidentally create duplicate records. Tables also cannot be loaded more than once, because the unique index prevents the loading of duplicate records. The primary index allows for common indexed access of tables from business logic.

**Examples**

**Right**

**Design**

Assume that a new contact table, adcd_det is being added to qaddb in eB3 to store contact details for 0 to n contacts for each address (ad_mstr record).

The main OID field is named oid_adcd_det and because this table is in qaddb, a main OID index will be added that is unique and will be named oid_adcd_det.

And because this table will be a child of the ad_mstr table, a foreign key OID field named oid_ad_mstr will also be added. The most common business logic access to the table will be through a join with the ad_mstr table. Therefore, the primary business index will consist of the foreign key OID field (oid_ad_mstr) plus a field for sorting, adcd_contact_name. We also want to ensure that two contacts with the same name are not added for one address, so this primary index will also be unique. And because this primary index starts with a foreign key OID field, it will be named adcd_oid_ad_mstr to comply with STD-0325: Use OID fields as foreign keys to relate tables (eB3 and up)

**Code**
Wrong Design

The name of the main OID index below does not match the main OID field name. The main OID index should not start with the table prefix. Also, this index is not the primary access through business logic and therefore should not be primary.

Code
ADD TABLE "adcd_det"
AREA "Schema Area"
LABEL "Address Contact Detail"
DESCRIPTION "Address Contact Detail"
DUMP-NAME "adcd_det"
TABLE-TRIGGER "Create" NO-OVERRIDE PROCEDURE "adcdc.t" CRC "?"

ADD FIELD "oid_adcd_det" OF "adcd_det" AS decimal
DESCRIPTION "Unique application generated record identifier."
FORMAT ">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>9.9<<<<<<<<<<<<" INITIAL "0"
MAX-WIDTH 38 DECIMALS 10 MANDATORY

ADD FIELD "adcd_addr" OF "adcd_det" AS decimal
DESCRIPTION "The unique code that identifies an address."
FORMAT "x(8)" INITIAL "" LABEL "Address"
MAX-WIDTH 80 MANDATORY

ADD FIELD "adcd_contact_name" OF "adcd_det" AS character
DESCRIPTION "Name of contact."
FORMAT "x(35)" INITIAL "" LABEL "Contact Name"
MAX-WIDTH 35 MANDATORY

ADD INDEX "adcd_oid_adcd_det" ON "adcd_det" (Wrong name for main OID index) PRIMARY (Should not be primary.
Another primary index is needed)
UNIQUE
INDEX-FIELD "oid_adcd_det" ASCENDING

See Also

STD-0286 Index Names
DST-0005 Developing with OID fields (eB3 and up)
STD-0325 Use OID fields as foreign keys to relate qaddb tables (eB3 and up)
STD-0050 Required Table Fields
STD-0061 Make all fields 'Mandatory'
STD-0007 UNKNOWN values used in Unique Keys in Indexes Must Be Unique
STD-0180 Every table must have a unique index & a primary index (pre-eB3)
STD-0328 All field names must be qualified in LIKE clauses in mfdeclre.i

Summary

When defining variables in mfdeclre.i, always qualify field names used in LIKE clauses with the table name.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0328</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress,Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>MFG/PRO Application,Variable Definition</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Because mfdeclre.i is sometimes referenced in schema trigger source files, all variable definitions that use the 'LIKE field_name' in mfdeclre.i must qualify the field name with its table name. This is because when write trigger files execute, Progress automatically defines two buffers for the table record ('before' and 'after' versions of the record). If the field name is not qualified in the variable definitions using the

7. mfdeclre.i must qualify all database field names in LIKE clauses

mfdeclre.i must be included inside all schema trigger programs, as the values of MFG/PRO global variables required to execute gprun.i, etc., are otherwise not available to the trigger programs. Because the 'before' and 'after' versions of the trigger's associated table are provided to the program automatically by PROGRESS through special buffers, all mfdeclre.i references to database fields inside LIKE clauses must be qualified using the table name in order to prevent compile errors. The database name should not be included in the qualification (e.g., qaddb.pt_mstr.pt_part), because this practice could lead to errors when running with multiple connected databases, such as Centralized SO Processing.
STD-0333 Database fields with Extents are not supported

Summary

No new fields that include multiple extents (Array datatypes) should be defined in any QAD product, especially the MFG/PRO or QAD Financials databases.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0333</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Draft</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Fields, Table Access &amp; Storage</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

Effective with this standard, no new schema fields that include extents (array fields) are allowed to be added to the data dictionary of any QAD product. If multiple occurrences of the same data element are required, the schema design should include either multiple similar fields, or a child table with multiple entries - whichever approach is more appropriate for the requirement.

Rationale

Array / Extent fields have caused issues in the past in the areas of Browse Definitions, Scrolling Windows, Oracle and SQL Server DB conversion, etc. Also, ODBC clients have difficulty accessing data in Progress array field extents. Lastly, schema fields with extents are not supported in the Component Builder which is used for the development of the QAD Financials application.

Additional Information from Progress:

In a relational database, the use of array fields is not recommended. In fact, it is best never to use arrays in your database definition. You might think that using array fields increases performance by reducing the number of related records needed to calculate some information. However, you are likely miscalculating the actual cost of using this technique:

- Consider that array fields are inherently non-normalized — they attempt to represent a one-to-many relationship in a single record. Arriving at the appropriate number of extents for an array is often an arbitrary decision. If the original number of extents turns out later to be insufficient, you must make a schema change, and probably also change your application code, to deal with this issue.
- Arrays can lead to large records that span multiple database blocks. This increases the number of reads required to retrieve a single record and reduces any performance gained by using them. Transporting such large records across a network requires increased bandwidth and decreases performance.
- Array values cannot be efficiently indexed, cannot be used as a join field, and cannot be accessed using SQL syntax (for example, from a reporting tool that will want to report on the data those array fields hold). You cannot filter array values; that is, you cannot write a single query statement to retrieve records where one of the values in an array field matches a filtering condition. In other words, you cannot manipulate the data on the client, sort or filter on it, or report on it.
- Standard Progress Dynamics components also cannot display array data.

Examples

Wrong

Code
ADD FIELD "my_array_fld" OF "my_det" AS character
DESCRIPTION "Array field with 100 extents"
FORMAT "x(99)"
INITIAL ""
LABEL "Field Value"
EXTENT 100
MAX-WIDTH 10100

See Also

STD-0006 Allowed number of fields per table
STD-0050 Required Table Fields
STD-0051 Providing for Extra Customer Fields
STD-0307 Providing for Extra QAD Fields
STD-0351 Supported Progress Data Types
STD-0336 Unicode & Code Page-independent Programming Issues with LENGTH, SUBSTRING & OVERLAY

Summary

In Progress, the LENGTH, SUBSTRING functions and the OVERLAY statement can all include a type parameter that directs Progress to interpret the specified position and length values as character units, bytes, or columns. There are FOUR valid types: CHARACTER, FIXED, COLUMN, and RAW. Misusing the type parameter will cause portability and code page compatibility issues in the program, making it only compatible with some code pages. This document provides guidelines as to how to use the appropriate type parameter to write code page-independent programs.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0336</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Program Style</td>
</tr>
<tr>
<td>Applicability</td>
<td>As of eB3 with Open Edge 10.1A and Up.</td>
</tr>
</tbody>
</table>

Description

In Progress, the LENGTH, SUBSTRING functions and the OVERLAY statement can all include a type parameter that directs Progress to interpret the specified position and length values as character units. There are FOUR valid types: CHARACTER, FIXED, COLUMN, and RAW.

"CHARACTER" – The expression "CHARACTER" specifies character units.

"FIXED" (SUBSTRING only) – The expression "FIXED" specifies that position is in character units and the length is in bytes, but directs SUBSTRING to yield only whole characters.

"COLUMN" – The expression "COLUMN" specifies display or print character-columns.

"RAW" – The expression "RAW" specifies bytes.

Each type above has its own measure units. To make programs code page-independent, you must use LENGTH, SUBSTRING and OVERLAY with appropriate types in different scenarios. Follow the guidelines below.

1. RAW should NOT be used for code page dependent operations. RAW means that character data is processed with byte units. LENGTH, SUBSTRING, and OVERLAY with RAW type parameter can be used to perform binary stream operations, such as I/O, file Import/Export, and so on. For single-byte code pages like ISO8859-1, you can use the three type parameters interchangeably with no impact on the code. However, for double-byte code pages like CP936 and Unicode (UTF-8), which use 1-N bytes to represent a character, do not use the RAW type in the abovementioned functions to display or manipulate data in such operations as indexing and extracting sub-strings because the Raw type results in different returned values depending on the code page being used. The following example uses Chinese data to demonstrate the impact of the RAW type on returned values of the LENGTH function:

```sql
DEFINE VARIABLE ch AS CHARACTER NO-UND0.
ch = "上海企业家".
Values of LENGTHS using different types:

<table>
<thead>
<tr>
<th>Codepage</th>
<th>length(ch,&quot;RAW&quot;)</th>
<th>length(ch,&quot;CHARACTER&quot;)</th>
<th>length(ch,&quot;COLUMN&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP936</td>
<td>10</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>UTF-8</td>
<td>15</td>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>
```

In addition, since UTF-8 is not a length-fixed code page, characters encoded with UTF-8 may have different byte lengths ranging from 1 to N. Here is an example by using SUBSTRING to extract the first character of the same data used in the previous example.
On balance, do not use the RAW type in the LENGTH & SUBSTRING functions to perform code page dependent operations, such as displaying data. It can be used to in code page-independent operations such as binary data I/O and byte stream processing.

2. CHARACTER is code page independent and should be used for character data processing not involving UI layout. CHARACTER is code page independent. No matter how many bytes a character in a code page is encoded with, it is treated as one character unit by LENGTH & SUBSTRING using CHARACTER. Therefore, we recommend you use CHARACTER to manipulate character data, like indexing, extracting a sub-string, and the code written in this way are code page transparent and portable.

On the other hand, because in some languages one character may occupy more than one display and print character-columns, CHARACTER should not be used by LENGTH & SUBSTRING which are involved in the logic of determining the layout of user interfaces.

For example, if LENGTH is using CHARACTER to calculate the display length of a double-byte language data, the returned length value will be only half of the actual display width of the data because one double-byte character needs two display and print character-columns.

Due to the above-mentioned reasons, the CHARACTER type should not be used in code involving user interface layout.

3. COLUMN is code page independent and should be used to deal with layout logic of user interfaces. As COLUMN specifies display or print character-columns of a character that is language-specific rather than code page-specific, it should be used by LENGTH, SUBSTRING, and OVERLAY in programs involving UI layout, such as displaying data in labels or arranging relative positions of UI components. In other words, the COLUMN type is more apt for viewing data than processing data.

For example, in the glexcnt.p program, the following statement should be changed from:

```c
excd_rate:label = fill(" ", 14 - length(c-trans-label, "RAW")) /* in eB2.1 */
```

to:

```c
excd_rate:label = fill(" ", 14 - length(c-trans-label, "COLUMN")) /* in eB3 */
```

4. Take care when using more than two types in one statement. Sometimes, one statement may contain LENGTH, SUBSTRING, and OVERLAY, or two of these functions, and it is possible that more than one type is used. In principle, it does not make sense to use multiple types and should be avoided in most circumstances.

For example, a statement in mfrtrail.i in eB2.1 reads like this:

```c
1_execname = substring(execname, 7, length(execname, "RAW"), "RAW").
```

Because the SUBSTRING function is designed to extract a sub-string and should use the CHARACTER type as much as possible in order to make the codes using it codepage-independent and portable. Therefore, the above statement should be changed to:

```c
1_execname = substring(execname, 7, length(execname, "CHARACTER"), "CHARACTER").
```

Now, it is no longer reasonable for LENGTH in this statement to work with "RAW", and this statement should be further changed to:

```c
1_execname = substring(execname, 7, length(execname, "CHARACTER"), "CHARACTER").
```

Here, the reason why we choose CHARACTER is because the purpose of this statement is to set a character data
to a character variable and this is a pure data processing and has nothing to do with view logic of data.

**Examples**

**Right**

**Code**

Example 1: FUNCTION getFieldLabel in gplabel.p in eB3

```plaintext
if available ( tt_lbl )
then do:
  if length(tt_lbl_long,"column") &lt;= p-length and
  tt_lbl_long &lt;&gt; ? and tt_lbl_long &lt;&gt; "" then
  c-label = tt_lbl_long.
else
  if length(tt_lbl_medium,"column") &lt;= p-length and
  tt_lbl_medium &lt;&gt; ? and tt_lbl_medium &lt;&gt; "" then
  c-label = tt_lbl_medium.
else
  if length(tt_lbl_short,"column") &lt;= p-length and
  tt_lbl_short &lt;&gt; ? and tt_lbl_short &lt;&gt; "" then
  c-label = tt_lbl_short.
else
  c-label = substring(tt_lbl_long,1,p-length,"column").
end.
```

Example 2: mfdtitle.i in eB3

```plaintext
if substring(dtitle,1,1) &gt;= "0" and substring(dtitle,1,1) &lt;= "9"
then dtitle = "{1}" + dtitle.
else dtitle = substring(dtitle,1,index(dtitle," ")+1)+
  substring(dtitle,index(dtitle," ")+length("{1}")+1,78).
```

Example 3: esgensup.p in both eB2.1 and eB3 (low-level code page independent operation)

```plaintext
if lc_source = "NEW" then
do:
  set-size(lm_ptr) = length(lc_new_stamp,"raw") + 1.
  put-string(lm_ptr,1) = lc_new_stamp.
end.
else
do:
  set-size(lm_ptr) = length(lc_old_stamp,"raw") + 1.
  put-string(lm_ptr,1) = lc_old_stamp.
end.
```

**Wrong**

**Code**
Example 4: FUNCTION getFieldLabel in gplabel.p in eB2.1
if available ( tt_lbl )
then do:
  if length(tt_lbl_long,"raw") &lt;= p-length and
      tt_lbl_long &lt;&gt; ? and tt_lbl_long &lt;&gt; ""
  then
    c-label = tt_lbl_long.
  else
    if length(tt_lbl_medium,"raw") &lt;= p-length and
       tt_lbl_medium &lt;&gt; ? and tt_lbl_medium &lt;&gt; ""
    then
      c-label = tt_lbl_medium.
    else
      if length(tt_lbl_short,"raw") &lt;= p-length and
         tt_lbl_short &lt;&gt; ? and tt_lbl_short &lt;&gt; ""
      then
        c-label = tt_lbl_short.
      else
        c-label = substring(tt_lbl_long,1,p-length,"raw").
  end.
end.

Example 5: mfdtitle.i in eB2.1
if global_usrc_right_hdr_disp &lt; 2 then do:
  if substring(dtitle,1,1) &gt;= "0" and substring(dtitle,1,1) &lt;= "9"
  then dtitle = "{1}"+ dtitle.
  else dtitle = substring(dtitle,1,index(dtitle," ") + 1,?8).
end.

See Also

N/A
STD-0337 Issue with TEXT() option in SET, UPDATE and ENABLE statements

Summary

Avoid using the TEXT() option in SET, UPDATE and ENABLE statements under Desktop/.NetUI.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0337</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Desktop UI Apps, Double-Byte Languages, Unicode</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

TEXT() option is used in data manipulation statements such as SET, UPDATE and ENABLE to automatically wrap the word that user inputs. The TEXT() option only works with character fields and does NOT SUPPORT Unicode code pages or Double-Byte code pages when running under Desktop/.NetUI. Developers must get around this TEXT() option under Desktop/.NetUI.

Rationale

If TEXT() option is necessary in a SET, UPDATE or ENABLE statement, put a condition to check if the session is running under Desktop/.NetUI or not before the SET, UPDATE or ENABLE statement. If it's running under Desktop/.NetUI, just do not use Text() option, since widget-walker works fine with or without this TEXT() option. If it's not running under Desktop/.NetUI, then use the Text() option.

Here's a flow diagram describes this logic:

```
Start

Running under Desktop/.NetUI

N

Code with Text()

Y

Code without Text()

Continue
```
Example

```haskell
if (gpiswrap.i) then /* Do not use text() under Desktop/.NetUI */
  Update
    ecm_date
    ecm_init
    ecm_cmmt
  GO-ON ("F5" "CTRL-D").
else
  Update
    ecm_date
    ecm_init
    Text (ecm_cmmt)
  GO-ON ("F5" "CTRL-D").

/* Use TEXT() option directly with no condition check */
Update
  ecm_date
  ecm_init
  Text (ecm_cmmt)
  GO-ON ("F5" "CTRL-D").
```

See Also

N/A
STD-0340 Proper assignment of alias for Progress and Oracle databases

Summary

All products that connect to the MFG/PRO database must properly assign the qaddb and qad aliases for the proper MFG/PRO database backend.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0340</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress,Oracle/SQL</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>DB Tables,MFG/PRO Application,Procedure Structure</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities</td>
</tr>
</tbody>
</table>

Description

In order to ensure proper connection to an Oracle database as well as the ability to share r-code, the database aliases must be assigned correctly. The primary MFG/PRO database must have the alias qaddb to enable shared r-code capabilities.

When running an application which uses an Oracle database for MFG/PRO, the Progress session first connects to a Progress database which contains the schema (schema holder) using the alias qad, then subsequently connects to the Oracle database using the alias qaddb. Progress cannot correctly assign a logical name to the Oracle database at startup (-ld qaddb), the alias assignment must therefore be enforced via code.

Although Progress databases can correctly assign the logical name at runtime, Progress MFG/PRO databases are still assigned the alias of qaddb via code. This safeguards runtime and r-code compatibility issues for Progress databases regardless of the logical name passed at database connection.

Additionally, because MFG/PRO code is compiled against a database with the logical database name of qaddb, this code must be executed before any other processing is performed against the database to make sure the qaddb alias is created. If not, the programs will fail at runtime because there is no database connected with the logical database name or alias that the compiled code requires.

The existing routine: mfcqa.p should be used to set the aliases for standard Progress client/server applications.

The 'CREATE ALIAS' statement should not be used in any new business logic.

For products that do not have a "true" client connection, use a wrapper routine to assign the alias prior to the execution of any application logic.

Rationale

All QAD Progress products that leverage the MFG/PRO application database, must work regardless of whether the database is Progress or Oracle.

Example
1) Logic from mf.p (Main Menu Program for MFG/PRO)

/*CHECK NUM-DBS AND SET QADDB, QAD ALIASES*/
do on error undo, leave:
  run mfcqa.p(input "").
  run mf1.p.
  return.
end.

2) Logic from mfaistrt.p (API start-up program for AppServer)

/*
*  CREATE DATABASE ALIASES, REQUIRED FOR ALL SUBSEQUENT
*  PROGRAMS TO RUN
*/
run mfcqa.p (input "").

/*
*  INSTANTIATE GLOBAL SESSION VARIABLES
*/
run pxdeclre.p.

Sample wrapper program:

{mfdecweb.i "NEW"}
{wbpp01.i}
{wbgp05.i}
{wbgp06.i}
run chk-db-and-create-alias no-error.
run mainprogram.p

Additional Information:
The internal procedure chk-db-and-create-alias is contained within the
include file {wbgp06.i}:

See Also

STD-0323 Switching domains in procedure code
STD-0145 Multiple DB - Switching aliases (pre-eB2.1)
STD-0341 Assigning correct value to 'hi_char' when code page is UTF-8 or collation is ICU

Summary

With the introduction of Unicode and its corresponding collation method called ICU, the traditional algorithm generating "hi_char" value no longer applies. New algorithm and considerations should be taken if "hi_char" is used in a Unicode environment.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0341</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Data Handling Logic, Unicode</td>
</tr>
<tr>
<td>Applicability</td>
<td>As of eB3 with Open Edge 10.1A and Up.</td>
</tr>
</tbody>
</table>

Description

Many QAD products use a variable called “hi-char” when implementing a From/To query. If a user does not input any search criteria in the "To" field, hi-char, which literally means character larger than any other character, is assigned to the "To" field so that the range of search result is from the value set in the "From" field to infinity. The traditional algorithm generating "hi_char" value is basically a bubble-sort algorithm sorting from code point 0 to 255 (0x00 to 0xFF). This range covers all the code points in a single byte code pages and works fine for double byte language with non-ICU collations since non-ICU collations will only collate based on the first byte. Sometimes, the "hi_char" value is merely set to a few character "Z" like hi_char="ZZZZZZ". This can only work for programs that deals ANSI characters only. However, With the introduction of Unicode and its corresponding collation method called ICU, the traditional algorithm generating "hi_char" value no longer applies. There are two problems in current traditional “hi-char” generation algorithm with UTF-8 and ICU. These problems even affects double byte code pages, if it is combined with ICU collation.

1. The bubble-sort algorithm is basically collating the current code point with the former one in a 0 to 255 loop and keeps the larger one. If Progress client is set to a non-UTF-8 code page with ICU collation, the collate operation converts characters to UTF-8 first and then does the comparison. In some code pages, some code point in 0 to 255 cannot be converted to UTF-8 and this might cause errors. The best way to solve this is to do a check in the loop before collate to see if the code point can be converted into UTF-8. If not, then skip the current iteration of the loop.

2. If Progress client is set to double byte code page with ICU collation, the search range of “hi-char” generation algorithm should be expanded to 2 bytes since ICU is not based on the first byte. The range should be 0 to 65535 (0x0000 to FFFF). If Progress client is set to UTF-8, the range should be even larger: 0 to 2^31-1 (0x0000 to 0x7FFFFFFF). Since sorting in such large ranges is quite time consuming, it’s better not to run the sorting algorithm on the fly during user login but put the result of these sorting in a data file.

To make the hi-char generation compliant with UTF-8 code page and ICU collation, the best practices are:

1. Keep the 0 to 255 sorting but add a UTF-8 conversion check before collating to avoid possible errors.
2. For clients set with ICU collation or UTF-8 code page, get “hi-char” value from a pre-calculated data file instead of generating it on the fly.

Examples

Right

Design

Add UTF-8 conversion check before collating in 0-255 search. Get hi_char from pre-calculated data file if clients set with UTF-8 codepage or ICU collation.

Code
Wrong

Design
Do a simple 0 to 255 search or just assign "ZZZZZZ" to hi_char.

Code

```java
do i = 0 to 255:
    if chr(i,"UTF-8")<>"" AND chr(i) > hi_char then hi_char = chr(i).

If cpinternal=“UTF-8” or cpcol1 begins “ICU” then
Set hi-char to chr (15572655) /*default hi_char value for UTF-8 code page
and ICU collation*/

If high-char data file exists then
Use the cpinternal value and cpcol1 value to fetch a hi-char value in the
temp table representing the data file. If no line matches, then do not set
the hi-char value, leave it as it was.
else
    show warning message “File "hi_char.dat" does not exist”,
    but still allow the program to continue, in this case, the default
    value will be used.
```

See Also

N/A
STD-0343 Utilities - Suggested program structure

Summary

Suggested utility program structure shown herein.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0343</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0</td>
</tr>
<tr>
<td>Language</td>
<td>Progress</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>Character UI Apps, Desktop UI Apps, Program Style, Reports/Inquiries</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development activities for MFG/PRO eB2.1 and up</td>
</tr>
</tbody>
</table>

Description

Decisions to be made regarding a utility that must be documented in the utility:

- Can it be run more than one time?
- Does the utility need a report?
- Would it be useful to have a simulation of what the utility will do before update mode is executed?
- When can this utility be obsoleted?

Program Naming Convention:

Use ‘ut’ as the first 2 letters of the program name when the utility can be run more than one time. ‘ut’ utilities are ones that can be used repeatedly, usually for correcting data that has somehow gotten out of sync or otherwise needs changing.

Use ‘ux’ as the first 2 letters of the program name when the utility should only be run one time for a domain, or one time if a system wide utility.

See STD-0310 Progress source code file names in MFG-PRO for additional source code name requirements.

When the utility is executed, the user should be presented with information about the utility that explains the purpose and processing notes. The text displayed to the user is stored in devmstr master comment data where:

- Master Reference = program file name (ex. uxfile.p)
- Language = “us”
- Type = “UT” (for both ‘ut’ and ‘ux’ utilities)
- Page = 1
- Text = Information presented to the user explaining its purpose and important processing notes

Use the following include file to present the comment data to the user:

```c
/* GET INFORMATIONAL TEXT using include file gpcdget.i for displaying to the user. */
gpcdget.i "UT"
```

If the utility is a ‘ux’ utility, create a mfc_ctrl record after the one time utility has been executed to identify that this utility should not be run again. This is typically setup so that the utility can be executed once for each domain. Alternatively, this can be setup as a system wide run, where the domain is blank.

Coding example:
/* CREATE RECORD TO INDICATE UTILITY HAS BEEN RUN */
if not can-find(first mfc_ctrl
    where mfc_domain = global_domain
    and mfc_field = "uxfile.p")
then do:
    /* XX = module */
    /* nn = number */
    /* "L" = logical */
    {mfctrlcr.i "uxfile.p" "" nn "L"}
end.

Optionally the utility can include a report, in which case the utility program must have the structure rules shown below.

The following are important reporting notes:

Please use mfrpchk.i instead of mfrpexit.i in all programs. While mfrpexit.i calls mfrpchk.i, we wish to eventually obsolete mfrpexit.i. mfrpchk.i cannot exist within the same block level as mfreset.i, mftrail.i, or mftr080.i. Putting it at the same level would cause the final include file to be skipped if an F4 is pressed or maximum pages is reached, and may also result in error messages such as "** terminal already has a conflicting use" or "Paged terminal streams may not be mixed with non paged". The final include file is necessary for page scrolling and for cleaning up after the report. See referenced Standard "Report - Page Limits and Printing Interruptions" on using mfrpchk.i if you have several levels within your report program.

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

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

Rationale

This ensures a standard look and feel for all utilities.

Example

Use the following template to design your utility program, where uxfile.p should be replaced with your utility program name.

```sql
/* uxfile.p - Description of utility */
/* Copyright 1986-2008 QAD Inc., Santa Barbara, CA, USA. */
/* All rights reserved worldwide. This is an unpublished work. */
/* */
/* PURPOSE: The purpose of this utility is to 
................. */
/* */
/* A ux*.p utility is run only one time 
................. */
/* A ux*.p utility can be run as many times as needed 
................. */
/* */
/* This utility produces a report showing 
................. */
/* This utility can optionally be run in simulation 
................. */
/* This utility can be obsoleted when 
................. */
```
/* $Revision: 1.2 $   BY: Employee Name    DATE: 08/22/08    ECO: *XXXX* */
/*-Revision
end--------------------------------------------------------------------------------*/

/*V8:ConvertMode=Report */

/* DISPLAY TITLE */
{mfdtitle.i "99 "}

/* PREPROCESSOR USED FOR REPORT'S WITH SIMULATION OPTION */
/* so that "SIMULATION" appears in the report heading */
&SCOPED-DEFINE simulation true

/* PROCESSING CONTROL VARIABLES */
define variable continue like mfc_logical initial yes label "Continue" no-undo.
define variable update_yn like mfc_logical label "Update" no-undo.

/* SELECTION CRITERIA VARIABLES */
define variable select1 ....
define variable select2 ....

/* WORK VARIABLES */

/* BUFFERS */
define buffer   bf-mfc_ctrl for mfc_ctrl.

/* SELECTION FRAME */
form
   continue colon 35
   update_yn colon 35 skip(1)
with frame a side-labels width 80.

/* SET EXTERNAL LABELS */
setFrameLabels(frame a:handle).

/* If a report is generated */
/* REPORT FRAME */
form
   field1
   field2
   ..... 
with frame rept down width 132.

/* SET EXTERNAL LABELS */
setFrameLabels(frame rept:handle).

/* CHECK IF UTILITY HAS ALREADY BEEN RUN */
if can-find(first mfc_ctrl
   where mfc_domain = global_domain
   and mfc_field = "uxfile.p")
then do:
   /* THIS UTILITY HAS ALREADY BEEN RUN */
   {pxmsg.i &MSGNUM=5127 &ERRORLEVEL=1}
   {gpwait.i}
   leave.
end.

/* THIS CODE READS A COMMENT MASTER RECORD WHERE: */

/* Master Reference = file name (uxfile.p) */
/* Language = "us" */
/* Type = "UT" */
/* Text has information presented to the user about this utility */
/* GET INFORMATIONAL TEXT */
gpcdget.i "UT"

/* CONTINUE? */
pxmsg.i &MSGNUM=2233 &ERRORLEVEL=1 &CONFIRM=continue

if not continue then
   return.

{wbrp01.i}
hide frame disp-cmmts.

repeat:
   /* RESET DISPLAY OF selection criteria */
   if select1 = hi_char then select1 = "".
   if effdate = low_date then effdate = ?.
   if effdate1 = hi_date then effdate1 = ?.

   update_yn = no.

   if c-application-mode &lt;"web" then
      update
         select1
         select2
         ....
         update_yn
      with frame a.

   {wbrp06.i &command = update
         &fields = "select1 select1 update_yn"
         &frm = "a"
   }

   if (c-application-mode &lt;"web") or
   (c-application-mode = "web" and
   (c-web-request begins "data")
   then do:
      bcdparm = "".
      {mfquoter.i select1}
      {mfquoter.i select2}
      {mfquoter.i update_yn}
   end.

   /* SET hi_char and low_date and hi_date for selection criteria */
   if select1 = "" then bill1 = hi_char.
   if effdate = ? then effdate = low_date.
   if effdate2 = ? then effdate2 = hi_date.

   /* OUTPUT DESTINATION SELECTION */
gpselout.i &printType = "printer"
   &printWidth = 132
   &pagedFlag = ""
   &stream = ""
   &appendToFile = ""
   &streamedOutputToTerminal = ""
   &withBatchOption = "yes"
   &displayStatementType = 1
   &withCancelMessage = "no"
   &pageBottomMargin = 6
   &withEmail = "yes"
QAD Development Standards

/* REPORT HEADER */
{mfphead.i}

/* DATA CONVERSION LOGIC */
for each .... no-lock where

display
   field1
   field2
   field3
with frame rept down width 132.

down 1 with frame rept.

field3 = new value. /* data conversion logic */

if update_yn then /* not in simulation mode, so update database */

   database_field = field3.

{mfrpchk.i}

end. /* for each .... */

/* REPORT TRAILER */
{mfrtrail.i}

if update_yn then do:
   do transaction:
      /* CREATE RECORD TO INDICATE UTILITY HAS BEEN RUN */
      if not can-find(first mfc_ctrl
         where mfc_domain = global_domain
         and mfc_field = "uxfile.p")
         then do:
            /* XX = module */
            /* nn = number */
            /* "L" = logical */
            {mfctrclr.i "uxfile.p" "XX" nn "L"}
         end.
      end. /* DO TRANSACTION */
   end.

end. /* REPEAT */

See Also

STD-0310 Progress source code file names in MFG-PRO
STD-0347 Source file formatting - Header of files in Subversion (SVN) eB3 and above

Summary

The header of all source code files controlled by Subversion, regardless of language, must conform to the format shown here.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0347</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>4GL/ABL/C/C++/C#/Java</td>
</tr>
<tr>
<td>Status</td>
<td>Published</td>
</tr>
<tr>
<td>Categories</td>
<td>QAD EE Source code</td>
</tr>
<tr>
<td>Applicability</td>
<td>All development Activities eB3 and up</td>
</tr>
</tbody>
</table>

Description

The header of every source code file controlled by Subversion, regardless of language, must be formatted exactly as shown below. The following notation conventions have been used:
- angle brackets <> indicate a placeholder for a mandatory item, which is to be replaced with the actual value (brackets should be removed)
- square brackets [] indicate a placeholder for an optional item, which is to be replaced with the actual value (brackets should be removed)
- $Id$ is a mandatory Subversion Keyword that will be replaced with File Revision Information.

All items within the header must be formatted as comments, and are shown here using the Progress comment delimiters /* */. If the programming language in use does not support this type of comment delimiter, appropriate comment delimiters should be substituted.

/* <Filename> - <Brief Description> */
/* Copyright 1986-<Year> QAD Inc., Santa Barbara, CA, USA. */
/* All rights reserved worldwide. This is an unpublished work. */
/* $Id: */
/* */
/* <Detailed Description> */
/* */
/* */

Each item shown above is described in detail below, in the order that it must appear in the header.

Filename - The filename of the source code file, including language-specific extension. Use mixed case. Appears on the same line as the filename.

Year - The current year in YYYY format, to be placed as shown in the official QAD copyright statement. This field must be updated (if not the current year) anytime a source file is checked in.

File Revision Information - based on Subversion svn:keywords "$Id:". This keyword automatically records last revision made to the file, including file name, revision, time stamp and author. The format is described below.

Detailed Description - A long description of the purpose of the source file, possibly multiple lines in length. Should be mixed case as in standard English. The format is described below.

Detailed Description

The detailed description consists of one or more lines of the form:

/* <Detailed Description Line> */
...

Where the place holder Detailed Description Line represents the line(s) containing the long description of the
purpose of the source file.

File Revision Information- $Id$

The file revision information consists of a single line and should only contain the "$Id::" Subversion keyword (svn:keywords Id property). This line is not to be copied over like in previous standards that would have the header information grow as changes are made. The file revision information will be replaced by Subversion data as follows:

/* $Id:: File name - Revision - UTC Date Time - User id :: */

The $Id:: is a special Subversion keyword that will get replaced with file name, revision, time stamp and user id as shown above, but for this to work it has to be "enabled" on a per file basis. This feature can only be used on text files (Progress ABL, Java, C, C++, C#, perl, jsp, html source files). Do not turn on this feature for binary files.

This line does not need manual edits to maintain the revision history on each file. That information can be extracted from Subversion. All that is needed is the last revision where a given file was modified and use the Subversion log functionality.

Enable File Revision Substitution (When adding new files only)

In Subversion the Id keyword is not turned on by default, this is enabled via file properties associated to each file. The following two steps are required only when adding a new file (Note that existing files will already contain the Id/svn:keywords enabled and no action is required):

1. Add the $Id: line below the "All rights reserved worldwide" line (usually 4th line of the file). The format of the line should be as follows

/* $Id:: */
$Id: */

The spacing is important and should align with the other header lines. From start to end of comment there should be 80 characters. When Subversion substitutes the Id information the lines are padded and spacing is preserved by using the above format.

2. Turn on the $Id keyword in Subversion prior to committing the new file. To do so, issue the Subversion command

svn propset svn:keywords "Id" <file name(s)>

In SVN Tortoise, right click on the folder/file and select "properties" specify svn:keywords and add Id as the property contents

What a developer needs to do to an existing source file header in Subversion

All of the QAD EE Source files in Subversion have the Id and svn:keyword property enabled. The Id line should NOT be modified by any developer. Subversion will handle updates to this line via the svn:keywords property. The developer only needs to worry about other header information like copyright, description, etc.

What the 'svn commit' will do to a source file header in Subversion

The file stored in the repository is not keyword expanded, it will just contain something like $Id:. Any file checked out from Subversion is updated to indicate what revision the file was last modified in. This behavior prevents unnecessary differences and eases the merge process. 'svn diff' and 'svn merge' will know to ignore these lines.

Rationale

The establishment of this format for file headers makes it easy to determine the purpose and revision for any given source file. To obtain the history of all changes in a file, a QAD user can refer to the Subversion repository that hosts the file. By referring to 'svn log' functionality and JIRA anyone can determine all changes done prior to the specified revision. In addition, conformance to this standard will make it possible for tools to parse and extract the information contained in the header for a variety of purposes. The standard copyright statement provides legal protection for the intellectual property of the company.

Examples
Description:
/* This is the initial start-up program for MFG/PRO. */
/* */
/* This program can be compiled against any database because it */
/* has no database references. Its sole purpose is to set an */
/* alias for the first database named in the start-up script so */
/* that the real programs (which have been compiled against the */
/* pseudo-database name qaddb) can run. */
STD-0350 Maximum Record (Row) Size Limit

Summary

Do not define tables having record sizes which exceed 32k.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0350</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>4GL/SQL/C/C++/C#/Java</td>
</tr>
<tr>
<td>Status</td>
<td>Draft</td>
</tr>
<tr>
<td>Categories</td>
<td>MFG/PRO Application, Variable Definition</td>
</tr>
<tr>
<td>Applicability</td>
<td>All QAD products</td>
</tr>
</tbody>
</table>

Description

Progress imposes a 32k size limit on its addressable record buffer space. This means that the actual length of data stored in a record in a Progress database is limited to 32k in size and, more importantly, a table's defined record length cannot exceed 32k when ported to Oracle or SQL Server. Fatal errors occur upon table access when these limits are violated.

A record's actual length is the sum of the lengths of all data values stored in all fields in a single table row.

A record's defined length is the sum of all FIELD_WIDTH or MAX-WIDTH clauses within a table. The maximum size allowed is actually a little less than 32,768 bytes due to row overhead required by Progress.

The following runtime errors occur when an Oracle table accessible to Progress exceeds the Progress DataServer's 32k addressable limit:

```plaintext
ORACLE row may exceed 32k — can't allocate buffer for table <table_name> (328)
(6447)
Insufficient memory to allocate buffer for ORACLE (5038)
```

Similar errors occur when the sum of all data value lengths in a Progress database record exceeds the 32k buffer size.

Both Progress and Oracle (or SQL Server) will encounter this problem, but at different times:

- Progress databases will have problems when the actual data record grows to near 32K in size, or is initially created at 32k or larger, even if the defined record length exceeds 32k. This is because space for the record is not allocated in Progress until it is needed. Therefore, it is possible to define Progress records larger than 32k and not encounter errors until an actual 32K row is created in a Progress DB.
- On Oracle and SQL Server databases, on the other hand, space is pre-allocated to the Progress record buffer based upon the table's defined record length regardless of the actual length of the data. When a record definition (sum of _field_.width or MAX-WIDTH attributes) exceeds 32K, it cannot fit into the Progress record buffer, so an immediate error is returned. The Progress DataServer issues errors immediately upon access attempts, such as when trying to read an Oracle or SQL Server table into memory to access its contents, or even when dumping an empty table. These errors occur even when a table's actual data length is less than 32k in each record.

Make sure the sum of all field widths (_field_.width or MAX-WIDTH attributes) within a table does not encroach upon the 32,768 byte limit, and ensure that the actual size of all data values that an application will attempt to store in a record also does not come close to 32,768 bytes.
**SQL Server Warning**

SQL Server cannot split records across database blocks. SQL Server will yield write errors when the actual data/row size exceeds the database block size even when the actual data record length is less than the defined record length. QAD products which will run on SQL Server should not create records longer than the database block size (typically, 8K).

**Rationale**

This standard ensures the database-independence of QAD applications and allows our products to successfully run on Progress, Oracle or SQL Server databases.

**Examples**

**Right**

**Design**

The sum of all field width (MAX-WIDTH) values in a table does not come close to or exceed 32,768 bytes, and the application does not attempt to create records larger than about 32,000 bytes.

**Wrong**

**Design**

A table's schema definition contains large records whose total field widths exceed 32k, or the application attempts to write records larger than about 32k in size.

**Code**

The following .df example shows how array fields (which are prohibited, anyway, but used here only to create a concise example) can easily cause record definitions to exceed the 32k Progress record buffer size. The same problem would occur if the table consisted of eight 4000 byte character fields (FORMAT "x(4000)") or any other combination of field sizes that total 32k or more.

```plaintext
ADD TABLE "my_det"
  AREA "Schema Area"
  DESCRIPTION "My Long Detail"
  DUMP-NAME "my_det"

ADD FIELD "my_big_fld" OF "my_det" AS character
  DESCRIPTION "Medium Field Array."
  FORMAT "x(99)"
  INITIAL ""
  LABEL "Field Value"
  EXTENT 100
  MAX-WIDTH 10100

ADD FIELD "my_huge_fld" OF "my_det" AS character
  DESCRIPTION "Very Large Field Array."
  FORMAT "x(99)"
  INITIAL ""
  LABEL "Field Value"
  EXTENT 300
  MAX-WIDTH 30300
```

**See Also**

STD-0006 Allowed number of fields per table
STD-0301 Oracle MAX-WIDTH (SQL-WIDTH) must be specified on new database fields
STD-0333 Database fields with Extents are not supported
STD-0351 Supported Progress Data Types

Summary

Progress field data types allowed in QAD products.

<table>
<thead>
<tr>
<th>ID</th>
<th>STD-0351</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>4GL</td>
</tr>
<tr>
<td>Status</td>
<td>Draft/Published/Obsolete</td>
</tr>
<tr>
<td>Categories</td>
<td>MFG/PRO Application,Variable Definition</td>
</tr>
<tr>
<td>Applicability</td>
<td>All Progress-based QAD products</td>
</tr>
</tbody>
</table>

Description

The following Progress data types are allowed in QAD products:

- character
- decimal
- integer
- int64
- logical
- date
- datetime
- BLOB
- CLOB
- raw

Rationale

The above Progress data types help make QAD products database-independent and easily portable to other database systems. These data types are supported by the Progress DataServers for Oracle and SQL Server. Data types not included in the above list are not supported.

Examples

Wrong

The following data types are unsupported.

Design

Progress Data types not allowed in QAD products:

- ARRAY (Fields having the EXTENT schema definition attribute)
- DATETIME-TZ

Code

Array/Extent example:

```
ADD FIELD "my_array_fld" OF "my_det" AS character
DESCRIPTION "Array field with 100 extents"
FORMAT "x(99)"
INITIAL ""
LABEL "Field Value"
EXTENT 100
MAX-WIDTH 10100
```
See Also

STD-0333 Database fields with Extents are not supported