OpenEdge Application Example

AutoEdge

Programming Standards & Naming Conventions

Contents

1 Introduction

This document outlines a set of Programming standards and naming conventions to be used on the OpenEdge Application Example application. These standards are a culmination of many other works including guidelines from education, John Sadd's whitepapers and the Developers Guide.

2 Why have standards?

Before outlining the standards themselves it is worth considering why they are needed in the first place.

Standards are important to programmers for a number of reasons:

- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
- If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

Encourage good practice

Development standards should reflect 'good practice' so that by simply keeping to the standards developers can improve the quality of the code they produce. This is especially important for less experienced developers.

Standards also provide a measure against which code can be reviewed (use of code reviews can have a major impact on product quality).

Must be used

Having development standards is of no benefit if they are not adhered to. Standards can be implemented more easily if developers accept them. A major factor in this is that they are not over-elaborate or obstructive.

3 Database tables

- Tables should have a descriptive name that identifies the purpose of the table.
- They should only use alphanumeric characters.
- The first letter of each word in the name should be upper case and multi-word field names concatenated, e.g. TestDrive.
- Dump file names should be meaningful and unique.

4 Database Fields

- Field names are to be descriptive and only use alphanumeric characters.
- The first letter of each word in the field name should be upper case and multi-word field names concatenated, e.g. EngineSize.

- The primary unique key in all tables should be an ID field of type Integer. The name should be composed by the table name and an upper-case 'ID' at the end, e.g. "OrderID".
- Field names should be prefixed by the table name, unless they are foreign key fields from other tables. So the "name" field in the car table should be Car.CarName
- Fields that represent the same value, should have the same name and format in each table they are used, e.g. if the key for the Customer table is CustomerID, it will be referred to as Customer.CustomerID in the customer table and Order.CustomerID in the order table.
- All date fields are to have the format "99/99/9999".
- All logical fields should, by default have a format of "Yes/No". However, if the field does not present a yes/no answer, but simply a choice between only two possible values (e.g. male/female) the format can reflect these values.
- Fields are always to be given labels and help text, even if it is thought unlikely that they will be displayed on a screen or report.
- Descriptions are to be filled in outlining the role of the field, unless the help text is sufficient.
- If the mandatory box is checked then the initial value must be set to unknown (?).
- Fields are not to be made case-sensitive.

- Fields are not to be given assign triggers.
- Fields are not to be given a number of extents.

5 Indexes

- All tables should have a single, numeric field that can be used to uniquely identify the row. This should be in a unique, primary index.
- Index names should be left meaningful.

- Unique indexes are prefixed with a lower case ‘u’, e.g. uCustomerId.
- The first letter of the first field in non-unique indexes will be lower case, e.g. shortName.
- The description field should be filled in to indicate why the index was created.

6 Sequences

- Sequence names should start ‘seq’ followed for the field that they are being used to generate a value for, e.g. seqCustomerId.

7 Database triggers

- Trigger code is to be separated from the rest of the application code into a directory called triggers.
- Trigger procedures should named after the table they are for with a suffix of ‘-*t*.p’ where *t* is the type, ‘c’, ‘d’ or ‘w’, e.g. the Customer table would have a create trigger called customer-c.p.
- In the data-dictionary triggers will be referenced by a relative path starting with triggers, e.g. triggers/ customer-c.p.
- All path and trigger names are to be lower case.
- Within the triggers themselves normal programming standards will apply.
- Create triggers will generally be restricted to assigning values to key fields from sequences.
- Delete triggers will be used to prevent deletions from happening, deleting related records, updating audit tables, and updating count or total fields if appropriate.
- Write triggers will be used for updating fields with system generated values (dates, derived amounts, etc.) and auditing.

- No user-interface code is to be placed in triggers. To stop a trigger running it will return error and a message. It will then be up to the procedure generating the transaction to output the message in the most appropriate way. For example the delete trigger for the Customer table may have the following check in it:

```
IF CAN-FIND(FIRST Order WHERE Order.CustomerId =
                              Customer.CustomerId NO-LOCK) THEN
  RETURN ERROR "Customer has orders and can not be deleted.".
```

8 Naming conventions and references

Case

- All 4GL statements, functions, attributes, methods, etc. to be in upper case.
- Database tables and fields should not be abbreviated.
- Preprocessor names are to be in upper case.
- References to database tables, fields, sequences and indexes will follow the case guidelines given earlier.
- Field names will always be prefixed with table names, but not database names. This applies to temp-tables as well as database tables.

Variables

- Variable names must only contain letters and numbers. No other characters are permitted.
- The variable name should then be meaningful and have the first letter of each word capitalised, e.g. iTotalCars.
- All variables are to have the following data-type prefixes:

Data type	Prefix
Buffer	b or just the table name
Character	c
Com-handle	ch
Date	t
Decimal	d
Handle	h
Integer	i
Logical	l

Data type	Prefix
Memptr	p
Raw	w
Recid	re
Rowid	r
Stream	s
Temp-table	tt or e for use in ProDataSets
Work-table	w
ProDataSet	ds
DataSet-Handle	dh
Before-Table	<temp-table>Before

- Variables that are declared in the main block of a procedure (i.e. not in a user-defined function, trigger or internal procedure) have an additional prefix of ‘g’, e.g. gcOutputFile.

- Parameters follow the same conventions as variables, but with an additional prefix of ‘p’, e.g. pdInvoiceTotal.

Preprocessors & Include File Arguments

- Proprocessor names and named arguments to include files should follow the same naming conventions as variables, but are always prefixed with ‘x’, e.g.

```
&scop xiNumDays             7
&scop xcDayShortNames       "Mon,Tue,Wed,Thu,Fri,Sat,Sun":U
```

- Preprocessors and named arguments to include files never have the ‘p’ or ‘g’ prefix that variables can have.

- Preprocessors that are defined, but have no value (e.g. those designed to stop multiple inclusions of include files) should be defined as xDefFilename, for example:

```
/* Prevent multiple inclusions of this file. */
&IF DEFINED(xDefSystem) = 0 &THEN
  &GLOBAL-DEFINE xDefSystem
  ...
```

- Preprocessors and named arguments to include files that do not resolve to a particular data-types are prefixed with x only, e.g.

```
&scop xcTableList "Customer,Order":u
&scop xTables     Customer Order
```

Translation Attributes

- All strings that are not to be translated should have :U after them (particularly filenames, event names and names of event procedures). This includes null strings and formats.

Internal Procedures and User-Defined Functions

- Internal-procedure (IP) and user-defined function (UDF) names are only to use letters and numbers with the first letter in lower case and the first letter of subsequent words capitalised, e.g.

```
RUN loadCustomerDetails.
```

- When naming internal-procedures and user-defined functions use the format actionObject, e.g. hideWindow.

Operators

- Use symbols instead letters for operators, e.g. = instead of EQ, <> instead of NE, etc.

File and Path Names

- File names should consist of lower case letters and numbers only. Spaces are explicitly forbidden.

- They should have a maximum base length of 20 characters, plus a '.' and a one character extension.

- Files containing source code will follow the following conventions, i.e.

File type	Convention
Procedure, with a user-interface that can be edited with the AppBuilder's Section Editor	.w
Procedure that does not have a user-interface and/or cannot be edited with the AppBuilder's Section Editor.	.p
Include file	.i
OERI Temp-Table Include file	et<TableName>.i
OERI Business Entity	be<EntityName>.p
OERI DataSource	sc<TempTable>.p
OERI ProDataSet Include file	ds<EntityName>.i
OERI DataAccessObject	da<TableName>.p
OERI DataAccess Validate file	da<EntityName>validate.p

File type	Convention
OERI Business Entity Validate file	be<EntityName>validate.p

- A reference to an external file or resource should always use a relative path, never an absolute one.
- In path names a '/' should be used in preference to a '\'.

9 Programming

General Guidelines

- Developers should actively look for ways to re-use code. Code that can be used in more than one place should either go into procedures or include files. The former (whether a super-procedure, an IP in a persistent library or an external .p) is preferable and should be used if at all possible.
- Variables, buffers, etc. should be declared in the tops of the triggers, IPs or UDFs, unless they are required to keep their values from one call to another.
- Do not put constants into code, always use a pre-processor instead, e.g.

```
&scop xiNumNames 10

DEFINE VARIABLE cNames AS CHARACTER EXTENT {&xiNumNames} NO-UNDO.

DO i = 1 TO {&xiNumNames}:
  ...
END.
```

- All variables, parameters, temporary-tables and work tables to be defined as NO-UNDO. If they do need to be undone if a transaction backs out a comment to this effect should be written next to the definition.
- All procedures with a user-interface should be capable of running persistently, unless they create a dialog-box.
- Parameters should not be passed to persistent procedures. Once loaded its IPs or UDFs should be called to pass data to it.
- Excessive applying of events should be avoided.
- At no time should program flow be dependant on labels (they will change during translation).

Code Blocks

- Internal procedures to be terminated with 'END PROCEDURE.'.

- CASE statements to be terminated with 'END CASE.'.
- REPEAT and FOR EACH block headers to be terminated with a ':'.
- Block labels are to be in upper case.
- Blank lines should be used to group statements that logically go together, not to separate every single statement.
- DO blocks should not be used to group 'logically' related statements together simply for cosmetic reasons.
- The END statement that terminates REPEAT, FOR EACH, DO blocks, etc. should be indented to the same level of the corresponding block header, e.g.

```
FOR EACH  ... :
  REPEAT ... :
  ...
  END.
END.
```

Code layout

- Code inside IPs, UDFs, triggers, code blocks and conditional statements to be indented two spaces. Other code will not be indented (code directly in the Definitions section or Main Block of a .w file).
- if statements are to be indented as follows:

```
IF <condition> THEN
  <single statement>
ELSE <single statement>

IF <condition> THEN
DO:
  <multiple statements>
END.
ELSE
DO:
  <multiple statements>
END.
```

- CASE statements are to be indented as shown below (also shows indenting within an IP):

```
PROCEDURE openQuery :
  DEFINE INPUT PARAMETER pcQueryType AS CHARACTER NO-UNDO.

  CASE pcQueryType:

    WHEN <condition > THEN
      <single statement>

    WHEN <condition> THEN
    DO:
      <multiple statements>
    END.

  END CASE.

END PROCEDURE.
```

- ASSIGN statements should have their first argument on the same line as the assign and subsequent one on succeeding lines, but in the same column as the first one, for example:

```
  ASSIGN cCustomerName  = fiCustomerName:SCREEN-VALUE
         cCustomerTelNo = fiTelNo.
```

- Each include file argument should go onto a new line, e.g.:

```
  {src/calc.i &InitValue = 3
         &ResultVar = iNewValue }
```

Comments

- All source files should have a standard header comment. For those files generated by the AppBuilder or Eclipse, the standard header should be modified. (A modified Eclipse Template is available upon request)

```
/* ------------------------------------------------------------------------------
  File        :
  Purpose     :
  Syntax      :
  Description :
  Created     :
  Notes       :

 Copyright (C) 2005 by Progress Software Corporation ("PSC"),
 14 Oak Park, Bedford, MA 01730, and other contributors as listed.

 All Rights Reserved.

 The Initial Developer of the Original Code is PSC.

 Software distributed under the License is distributed on an "AS IS"
 basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. You
 should refer to the License for the specific language governing
 rights and limitations under the License.
 ------------------------------------------------------------------------------*/
```

- Comments should be meaningful and add to the reader's understanding of why a particular section of code works like it does. It should not simply repeat the 4GL statements in another way.

- Internal Procedure & Functions should contain a standard header comment directly after the Procedure/Function declaration. To be consistent with AppBuilder generated Procedures/Functions it should take the following form:

```
/*------------------------------------------------------------------------
  Purpose:     Summary of the Procedure/Function
  Parameters:  Parameter Description
  Notes:       General notes
------------------------------------------------------------------------*/
```

- Comments should be indented to the same level as the code they refer to in the following manner, and should precede the code:

```
IF <condition> THEN
DO:
  /* This is a multi-line comment that refers to the multiple
    statements that are carried out if the above condition
    is true. */
  <multiple statements>
END.
```

- When amending code, the comments should be amended as well.

- Where a suitable software configuration management system (such as Roundtable) is being used obsolete code should be removed, not simply commented out.

- Where code is written to get round a 'feature' that is expected to be removed in a future release of Progress it should be commented as such. The comments must include the Progress version with which the code was written.

Reading Records

- Always explicitly state the lock status to be used when retrieving records.

- Use field lists on FOR EACH and DEFINE QUERY statements.

- ROWID should always be used instead of RECID.

Line Length

Avoid lines longer than 80 characters, since they're not handled well by many terminals and tools.

Note: Examples for use in documentation should have a shorter line length-generally no more than 70 characters.

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.
- Align the new line with the beginning of the expression at the same level on the previous line.

If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead.

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.

10 Restricted constructs

- No shared variable or other shared objects. Global shared items to be created where absolutely necessary.
- No WAIT-FOR statements are to be used in interactive code, apart from those automatically generated by the AppBuilder. Similarly conditional processing around the AppBuilder generated WAIT-FOR statements not to be removed.
- No INSERT, UPDATE, SET or READKEY statements are to be used in interactive code. Also editing-blocks are not to be used.
- RECIDs are not to be used unless interfacing to legacy code.
- Only NO-LOCK and EXCLUSIVE-LOCK to be used.

11 Modelling

As part of the project it is expected that UML models will be utilized throughout the whole SDLC process. In order to maintain standard terminology the base Enterprise Architect project must be used as a starting point for any modelling.

When modelling the following guidelines should be observed:

- Class/Component names should match the external progress procedure name.
- When applicable the language property should be set to 'Progress'.
- When applicable the provided Progress aware Stereotypes should be used.