IBM

ESQL code conventions in WebSphere Message Broker

Rachel Shen (rshen@ca.ibm.com), Certified Senior IT Specialist, IBM
Ankur Upadhyaya (ankur@ca.ibm.com), IT Specialist, IBM

Summary: These coding conventions for Extended Structured Query Language (ESQL) will help you develop ESQL code that is easy to understand and maintain. This article is for developers using ESQL to implement message flow applications for deployment on WebSphere Message Broker.

Date: 12 Mar 2008
Level: Intermediate
Activity: 1724 views

1. Introduction

The aesthetics and consistency of source code determines if a program is easy to understand and maintain, which is critical to software development because 80% of its lifetime cost lies in its maintenance and enhancement by many different developers over time. Coding conventions not only help improve code readability, they also discourage the use of wasteful and error-prone programming practices. Coding guidelines can also encourage the development of secure software that has better performance. Finally, should source code need to be shipped as a product, coding conventions can help ensure a quality in presentation.

This article describes coding standards for Extended Structured Query Language (ESQL), emphasizing the use of ESQL in the development of IBM® WebSphere® Message Broker message flow applications. Topics include file naming and organization, file layout, comments, line wrapping, alignment, white space, naming conventions, frequently used statements, and useful programming practices.

2. File naming and organization

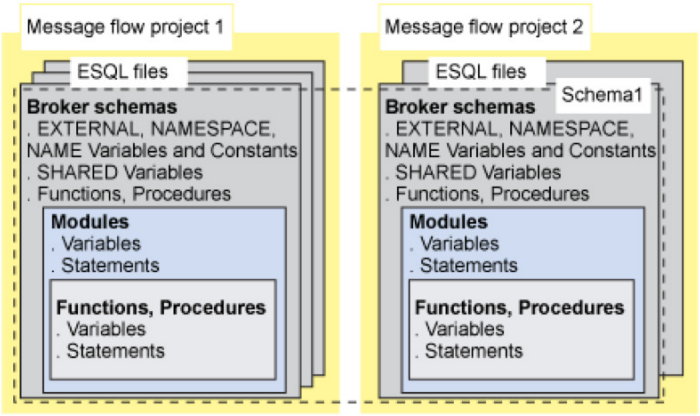
The following guidelines should be used when constructing the ESQL files that implement a WebSphere Message Broker application:

- ESQL source files should always have names that end with the extension .esql. For example: NotificationTimeout.esql.
- ESQL filenames should consist of mixed-case alphabetic characters, with the first letter of each word and all acronyms in uppercase. For example: IBMExample.esql.

In general, ESQL files longer than 2000 lines are cumbersome to deal with and should be avoided, by ensuring that a single ESQL file implements the message flows that relate to each other, and by abstracting reusable code into separate ESQL files.

ESQL files can be grouped in broker schemas, which are a hierarchical way of organizing ESQL files. They also serve to create local name spaces so that procedures and functions can be reused, and yet be distinguished by the schema they are in. In short, broker schemas are organizational units of related code that address a specific business or logical problem. Therefore, related ESQL files should be placed in their own schema.

Figure 1. ESQL file organization and layout



3. File layout

The content of each ESQL file should conform to the following standards:

- The file must start with a descriptive header comment, as described in Section 5 below.
- The header comment should be followed by a broker schema declaration and the PATH clauses that specify a list of additional schemas to be searched when matching function and procedure calls to their implementations. Do not use the default broker schema.

```
BROKER SCHEMA com.ibm.convention.sample
PATH com.ibm.convention.common;
PATH com.ibm.convention.detail;
```

The remainder of the file should be divided into three sections:

1. **Broker schema level variables and constants**

   They are not globally reusable and can only be used within the same broker schema.

   EXTERNAL variables are also known as user-defined properties. They are implicitly constant. When you use the NAMESPACE and NAME clauses, their values are implicitly constant and of type CHARACTER.

```
DECLARE DAY1 EXTERNAL CHARACTER 'monday';
DECLARE XMLSCHEMA_INSTANCE NAMESPACE 'http://www.w3.org/2001/XMLSchema-instance';
DECLARE HIGH_PRIORITY CONSTANT INTEGER 7;
DECLARE processIdCounter SHARED INTEGER 0;
```

2. **Broker schema level procedures and functions**

   They are globally reusable and can be called by other functions or procedures in ESQL files within any schema defined in the same or another project.

```
CREATE PROCEDURE getProcessId(OUT processId CHARACTER)
BEGIN
    BEGIN ATOMIC
        SET processId = CAST(CURRENT_TIMESTAMP AS CHARACTER FORMAT 'ddHHmmss')
                        || CAST(processIdCounter as CHAR);
        SET processIdCounter = processIdCounter + 1;
    END;
END;
```

```
CREATE FUNCTION encodeInBASE64(IN data BLOB)
RETURNS CHARACTER
LANGUAGE JAVA
EXTERNAL NAME "com.ibm.broker.util.base64.encode";
```

3. Modules

A module defines a specific behavior for a message flow node. It must begin with the CREATE node_type MODULE statement and end with an END MODULE statement. The node_type must be either COMPUTE, DATABASE, or FILTER. A module must contain the function Main(), which is the entry point for the module. The constants, variables, functions, and procedures declared within the module can be used only within the module.

```
CREATE FILTER MODULE FilterData
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        DECLARE messageType REFERENCE TO Root.XMLNSC.Msg.Type;
        IF messageType = 'P' THEN
            RETURN TRUE;
        ELSE
            IF messageType = 'D' THEN
                RETURN FALSE;
            ELSE
                RETURN UNKNOWN;
            END IF;
        END IF;
    END;
END MODULE;
```

4. Naming conventions

In general, the names assigned to various ESQL constructs should be mnemonic and descriptive, using whole words and avoiding confusing acronyms and abbreviations. Of course, you need to balance descriptiveness with brevity, because overly long names are hard to handle and make code harder to understand. The following table provides naming conventions for ESQL broker schemas, modules, keywords, correlation names, procedures, functions, variables, and constants.

Entity	Naming rules	Examples
Schema	A schema name reflects the file system name leading the location of files in this schema. Each level in a schema name is mapped to a directory name. To be supported by any file system, schema names should consist of lower case alphanumeric characters. The names should be prefixed with the reverse of the company URL.	com.ibm.convention.sample
Module	A module name should consist of more than one alphanumeric character, start with an upper case letter, and have mixed case, with the first letter of each internal word and all letters of acronyms in uppercase. It must match any label assigned to a compute, database, or filter node in a message flow that uses the module. If more than one such node in a message flow uses the same module, add additional characters to the node labels in order to differentiate between them.	ConstructInvoice1 ConstructInvoice2 IsReconnect RetrieveIBMData
ESQL keyword	ESQL keywords should be uppercase, although they are not case sensitive.	SET CREATE PROCEDURE TRUE
Field reference or correlation	A field reference or correlation name should start with an uppercase letter and have mixed case, with the first letter of each internal word and all letters of acronyms in uppercase.	Environment.Variables. Invoice. CustomerNumber

correlation name	uppercase.	CustomerNumber
Variable	A variable name should start with a lowercase letter and have mixed case, with the first letter of each internal word and all letters of acronyms in uppercase. Since a variable name should start with a lowercase letter, it should not start with an acronym. Trivial variable names such as i or x can be assigned to temporary variables of limited scope and importance at your discretion.	i invoiceItem currentHL7Section controlReference
Procedure or function	A procedure or function name should consist of more than one alphanumeric character, start with a lowercase letter, and have mixed case, with the first letter of each internal word and all letters of acronyms in uppercase. The first word of the name should be a verb.	setEnvironment computeIBMValue
Constant	A constant name should start with a letter, use all uppercase letters, and use the underscore (_) to separate words.	MIN_VOLUME MAX_RETRIES

5. Comments

The discussion below classifies ESQL comments into one of two classes:

- Header comments, used to summarize and demarcate a section of an ESQL file
- Implementation comments, used to clarify the meaning of a piece of ESQL logic.

5.1. Header comments

5.1.1. File header

An ESQL file should always begin with a file-header comment that provides the name of the file, a brief synopsis of the purpose of the file, the copyright, and author information. The example below illustrates one possible format for such a header, but any suitable alternative that clearly conveys the same information is acceptable. The header is 80 characters in length. The description text should consist of complete sentences, wrapped as needed without using hyphenation. List each author on a separate line.

```
/*
 *
 * File name: Workfile.esql
 *
 * Purpose:   Sample ESQL file with proper prologue.
 *
 * Authors:   Rachel Shen
 *                 Ankur Upadhyaya
 * Date      21 March 2008
 * Version:  1.0
 *
 * @copyright  IBM Canada Ltd. 2008.  All rights reserved.
 *
 */
```

5.1.2. Module header

Every module definition must be preceded by a module header comment. A module header contains descriptive text that consists of complete sentences, wrapped as needed without using hyphenation:

```
/*
 * Module description goes here
 *
 */
```

5.1.3. Procedure header

Every procedure definition must be preceded by a procedure header comment. A procedure header contains descriptive text that consists of complete sentences, wrapped as needed without using hyphenation. This header should also name and describe each of the parameters handled by the procedure, classifying them as type IN, OUT, or INOUT. The parameter descriptions need not consist of complete sentences -- brief, descriptive phrases should suffice. However, they should be preceded by a hyphen and properly aligned with one another:

```
/*
 * Procedure description goes here.
 *
 * Parameters:
 *
 * IN:     REFERENCE parameter1 - Description goes here.
 * INOUT: INTEGER   parameter2 - Description goes here.
 * OUT:    TIMESTAMP result     - Description goes here.
 *
 */
```

5.1.5. Function header

Function headers are essentially the same as procedure headers:

```
/*
 * Function description goes here.
 *
 * Parameters:
 *
 * IN:     REFERENCE parameter1 - Description goes here.
 * INOUT: INTEGER   parameter2 - Description goes here.
 * OUT:    TIMESTAMP result     - Description goes here.
 *
 * RETURNS: BOOLEAN â€" Description goes here.
 *
 */
```

5.2. Implementation comments

Add comments to ESQL source code to clarify program logic and convey information that is not immediately obvious from inspecting the code. Do not add too many comments -- they can become redundant, complicate code maintenance, and get out of date as the software evolves. In general, too many comments indicate poorly written code, because well written code tends to be self explanatory. Implementation comments can be written in single-line, block, or trailing forms, as described below.

5.2.1. Single-line comments

A single-line comment is a short comment that explains and aligns with the code that follows it. It should be preceded by a single blank line and immediately followed by the code that it describes:

```
-- Check for the condition
IF condition THEN
    SET z = x + y;
END IF;
```

5.2.2. Block comments

Block comments are used to provide descriptions of ESQL files, modules, procedures, and functions. Use Block comments at the beginning of each file and before each module, procedure, and function. You can also use them anywhere in an ESQL file. Block comments inside a function or procedure should be indented at the same level as the code they describe. Precede a block comment by a single blank line and immediately follow it by the code it describes. Because shorter comments with expressive code are always preferable to lengthy block comments, block comments should be rarely used within a procedure or function. Example of a block comment:

```
/*
 * Here is a block comment to show an example of comments
 * within a procedure or function.
 *
 */
IF condition THEN
    SET z = x + y;
END IF;
```

5.2.3. Trailing comments

Trailing comments are brief remarks on the same line as the code they refer to. Indent these comments to clearly separate them from the relevant code. If several trailing comments relate to the same segment of code, align the comments with one another. Trailing comments are usually brief phrases and need not be complete sentences:

```
IF condition THEN
    SET z = x + y;           -- trailing comment 1
ELSE
    SET z = (x - y) * k;     -- comment 2, aligned with comment 1
END IF;
```

6. Style guidelines

6.1. Line wrapping and alignment

Lines of ESQL source code should be wrapped and aligned according to the following guidelines:

- Every statement should be placed on a separate line.
- Lines longer than 80 characters should be avoided as they exceed the default width of many terminals and development tools.
- The unit of indentation used to align ESQL source code should be four characters which is the default setting in the WebSphere Message Broker Toolkit. The specific construction of this indentation, using spaces or tabs, is left to the discretion of the programmer.
- Line lengths should be limited by breaking lengthy expressions according to the following rules:

- Lines should be as long as possible, without exceeding 80 characters;
- Break after a comma;
- Break before an operator;
- Break at the highest level possible in the expression;
- Align a new line with the beginning of the expression at the same level on the preceding line. Should this alignment require deep indentations that produce awkward code, an indentation of eight spaces may be used instead.

The following ESQL code samples illustrate the above rules.

```
CREATE FUNCTION function1(IN longExpression1 REFERENCE,
```

```
                          IN longExpression2 CHAR,
                          OUT longExpression3 NUMBER,
                          INOUT longExpression4 CHAR)
CALL procedure1(myLongVariable1, myLongVariable2, myLongVariable3,
         myLongVariable4);
SET returnValue = function1(argument1, argument2, argument3,
                            function2(argument4, argument5,
       argument6));
SET finalResult = ((x / variable1) * (variable2 - variable3))
               + (y * variable5);
```

The ESQL sample below illustrates a case where an indentation of eight spaces should be used instead of the usual alignment to avoid deeper indentations that would result in confusing code.

```
-- INDENT 8 SPACES TO AVOID VERY DEEP INDENT
CREATE PROCEDURE showAVeryLongProcedureName(IN argument1,
        INOUT argument2,
        OUT argument3)
```

Finally, the ESQL samples below illustrate line wrapping and alignment practices. The first is preferable because the line break is inserted at the highest level possible.

```
SET longName1 = longName2 * (longName3 + longName4 - longName5)
                  + 4 * longname6;                          -- PREFER
SET longName1 = longName2 * (longName3 + longName4
                  - longName5) + 4 * longname6;          -- AVOID
```

6.2. White space

White space should be used to improve code readability.

- Insert two blank lines between sections of a ESQL file;
- Insert one blank line between functions and procedures;
- Insert one blank line between the variable declarations in a function/procedure and its first statement;
- Insert one blank line before a block or single-line comment;

- A blank space should follow each comma in any ESQL statement that makes use of commas outside of a string literal;
- All binary operators should be separated from their operands by spaces.

```
SET a = c + d;
SET a = (a + b) / (c * d);
```

7. Statements

Each line should contain at most one statement. Here are some sample statements.

7.1. DECLARE

Put declarations right after the broker schemas or at the beginning of modules, functions, and procedures. One declaration per line is recommended:

```
-- EXTERNAL variable
DECLARE DAY1 EXTERNAL CHARACTER 'monday';
-- NAMESPACE variable
DECLARE XMLSCHEMA_INSTANCE NAMESPACE 'http://www.w3.org/2001/XMLSchema-instance';
-- CONSTANT
DECLARE HIGH_PRIORITY CONSTANT INTEGER 7;
-- SHARED variable
DECLARE processIdCounter SHARED INTEGER 0;
-- REFERENCE
DECLARE messageType REFERENCE TO Root.XMLNSC.Msg.Type;
```

7.2. FOR

```
DECLARE i INTEGER 1;
FOR source AS Environment.SourceData.Folder[] DO
    ...
    SET i = i + 1;
END FOR;
```

7.3. IF

```
-- IF statement
IF InputBody.Msg.Report = 'PDF' THEN
    SET OutputRoot.XMLNSC.Msg.Type = 'P';
END IF;
-- IF-ELSE statement
IF InputBody.Msg.Report = 'PDF' THEN
    SET OutputRoot.XMLNSC.Msg.Type = 'P';
ELSE
    SET OutputRoot.XMLNSC.Msg.Type = 'X';
END IF;
-- IF-ELSEIF-ELSE statement
IF InputBody.Msg.Report = 'PDF' THEN
    SET OutputRoot.XMLNSC.Msg.Type = 'P';
ELSEIF InputBody.Msg.Report = 'DOC' THEN
    SET OutputRoot.XMLNSC.Msg.Type = 'D';
ELSE
    SET OutputRoot.XMLNSC.Msg.Type = 'X';
END IF;
```

7.4. LOOP

```
DECLARE i INTEGER;
SET i = 1;
x : LOOP
    ...
    IF i >= 4 THEN
        LEAVE x;
    END IF;
    SET i = i + 1;
END LOOP x;
```

7.5. RETURN

```
RETURN;
```

```
RETURN TRUE;
RETURN FALSE;
RETURN UNKNOWN;
RETURN ((priceTotal / numItems) > 42);
```

7.6. THROW

```
THROW USER EXCEPTION;
THROW USER EXCEPTION CATALOG 'BIPv600' MESSAGE 2951
        VALUES('The SQL State: ', SQLSTATE, 'The SQL Code: ',
               SQLCODE, 'The SQLNATIVEERROR: ', SQLNATIVEERROR,
               'The SQL Error Text: ', SQLERRORTEXT);
```

7.7. WHILE

```
-- WHILE statement
DECLARE i INTEGER 1;
WHILE i <= 10 DO
    SET OutputRoot.XMLNSC.Msg.Count[i] = i;
    SET i = i + 1;
END WHILE;
```

7.8. CASE

```
-- CASE function
-- Like ?: Expression in C
SET OutputRoot.XMLNSC.Msg.Type = CASE InputBody.Msg.Report
        WHEN 'PDF' THEN 'P'
        WHEN 'DOC' THEN 'D'
        ELSE 'X'
        END;
-- Like SWITCH Expression in C
CASE InputBody.Msg.Report
WHEN 'PDF' THEN
    SET OutputRoot.XMLNSC.Msg.Type = 'P';
WHEN 'DOC' THEN
    CALL handleDocument();
ELSE
    CALL handleUnkown();
END CASE;
```

7.9. SELECT

```
SELECT ITEM segment.No
        FROM ControlRef.Segments.Segment[] AS segment
        WHERE segment.No = currentSegment;
```

7.10. UPDATE

```
UPDATE Database.TELEMETRY AS telemetry
        SET bitmap = refEnvTeleSeg.NewBitmap
        WHERE telemetry.TelemetryId = refEnvTeleSeg.Results.TelemetryId;
```

7.11. INSERT

```
INSERT INTO Database.TELEMETRY_SEGMENT (TelemetryId, BlockNum, FileSegment)
        VALUES (refEnvTeleSeg.Results.TelemetryId, RefEnvTeleSeg.SegmentNum,
                  ASBITSTREAM(Body));
```

8. Programming practices

- Put variables and constants at the broker schema level only when they need to be reused by multiple modules.
- Initialize variables within DECLARE statements, especially EXTERNAL variables.
- Declare REFERENCEs to avoid excess navigation of the Message Tree.
- Specify a direction indicator for all new routines of any type for documentation purposes although the direction indicator (IN, OUT, INOUT) of FUNCTION is optional for each parameter.
- Make code as concise as possible to restrict the number of statements. This will cut parsing overhead.
- Use LASTMOVE or CARDINALITY statements to check the existence of fields in the message tree. This would avoid mistakes.
- Avoid use of CARDINALITY statements inside loops.
- Avoid overuse of Compute nodes because tree copying is processor heavy: put reusable nodes into sub-flows.
- Avoid nested IF statements: use ELSEIF or CASE WHEN clauses to get quicker drop-out.
- Avoid overuse of string manipulation because it is processor heavy: use the REPLACE function in preference to a complete re-parsing.
- Use parentheses to make the meaning clear. The order of precedence in the arithmetic expressions is:

- Parentheses
- Unary operators including unary - and NOT
- Multiplication and division
- Concatenation
- Addition and subtraction
- Operations at the same level are evaluated from left to right.

9. Code sample

```
/*
 *
 * File Name: ESQLCodeConvention.esql
 *
 * Purpose:   Sample ESQL file with proper prologue.
 *
 * Authors:   Rachel Shen
 *                Ankur Upadhyaya
 * Date      21 March 2008
 * Version:  1.0
 *
 * @copyright  IBM Canada Ltd. 2008.  All rights reserved.
 *
 */
BROKER SCHEMA com.ibm.convention.sample
PATH com.ibm.convention.common;
-- First day of the week
DECLARE DAY1 EXTERNAL CHARACTER 'monday';
-- XML schema instance namespace
DECLARE XMLSCHEMA_INSTANCE NAMESPACE 'http://www.w3.org/2001/XMLSchema-instance';
-- High priority message's priority on MQ
DECLARE HIGH_PRIORITY CONSTANT INTEGER 7;
-- A shared counter to generate process id
```

```
DECLARE processIdCounter SHARED INTEGER 0;
/*
 * This procedure generates the process id.
 *
 * Parameters:
 *
 * OUT:   CHARACTER processId     - Description goes here.
 *
 */
CREATE PROCEDURE getProcessId(OUT processId CHARACTER)
BEGIN
    BEGIN ATOMIC
        SET processId = CAST(CURRENT_TIMESTAMP AS CHARACTER FORMAT 'ddHHmmss')
                        || CAST(processIdCounter as CHAR);
        SET processIdCounter = processIdCounter + 1;
    END;
END;
/*
 * This function encodes the input in BASE64 format.
 *
 * Parameters:
 * IN:    BLOB document - Input document for encoding.
 *
 * RETURNS: CHARACTER   â€“ BASE64 encoded string.
 *
 */
CREATE FUNCTION encodeInBASE64 (IN data BLOB)
RETURNS CHARACTER
LANGUAGE JAVA
EXTERNAL NAME "com.ibm.broker.util.base64.encode";
/*
 * This module has the sample code for the article, ESQL code convention.
 */
CREATE COMPUTE MODULE CreateESQLCodeConvention
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        DECLARE processId CHARACTER;  -- Unique identifier of the working process
        CALL getProcessId(processId);
        SET encodedMessage = base64Encode(ASBITSTREAM(InputBody.Msg));
        RETURN TRUE;
    END;
END MODULE;
/*
 * This module filters data base on the message type.
 * It has the sample code of the second module in the ESQL file.
 */
CREATE FILTER MODULE FilterData
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        DECLARE messageType REFERENCE TO Root.XMLNSC.Msg.Type;
        IF messageType = 'P' THEN
            RETURN TRUE;
        ELSEIF messageType = 'D' THEN
            RETURN FALSE;
        ELSE
            RETURN UNKNOWN;
        END IF;
    END;
END MODULE;
```

Acknowledgements

The authors would like to thank Steve Chai (chai@ca.ibm.com) for reviewing the article.

Resources

- WebSphere Message Broker developer resources page
  Technical resources to help you use WebSphere Message Broker for connectivity, universal data transformation, and enterprise-level integration of disparate services, applications, and platforms to power your SOA.

- WebSphere Message Broker product page
  Product descriptions, product news, training information, support information, and more.

- WebSphere Message Broker information center
  A single Eclipse-based Web portal to all WebSphere Message Broker V6 documentation, with conceptual, task, and reference information on installing, configuring, and using your WebSphere Message Broker environment.

- WebSphere Message Broker documentation library
  WebSphere Message Broker specifications and manuals.

- WebSphere Message Broker forum
  Get answers to your technical questions and share your expertise with other WebSphere Message Broker users.

- WebSphere Message Broker support page
  A searchable database of support problems and their solutions, plus downloads, fixes, problem tracking, and more.

- developerWorks WebSphere Business Integration zone
  For developers, access to WebSphere Business Integration how-to articles, downloads, tutorials, education, product info, and more.

- WebSphere Business Integration products page
  For both business and technical users, a handy overview of all WebSphere Business Integration products

- WebSphere forums
  Product-specific forums where you can get answers to your technical questions and share your expertise with other WebSphere users.

- Most popular WebSphere trial downloads
  No-charge trial downloads for key WebSphere products.

- Trial downloads for IBM software products
  No-charge trial downloads for selected IBM® DB2®, Lotus®, Rational®, Tivoli®, and WebSphere® products.

- Technical books from IBM Press
  Convenient online ordering through Barnes & Noble.

- developerWorks technical events and Webcasts
  Free technical sessions by IBM experts that can accelerate your learning curve and help you succeed in your most difficult software projects. Sessions range from one-hour Webcasts to half-day and full-day live sessions in cities worldwide.

About the authors

Rachel Shen is a Certified Senior IT Specialist in Application Integration and a member of the Americas IT Specialist Profession Board with IBM Canada. She has 14 years of application architecture and development experience in a variety of business sectors and technologies, including J2EE, Web services, and messaging. Rachel is part of the technical

team that provides ESB-based SOA solutions for a healthcare company using WebSphere Message Broker. You can contact Rachel at rshen@ca.ibm.com.

Ankur Upadhyaya is an IT Specialist with the IBM Pacific Development Centre in Burnaby, Canada. His areas of interest include all aspects of networking and distributed systems design. You can contact Ankur at ankur@ca.ibm.com.