IBM Software Group

Development, Test and Deployment Best Practices for WebSphere Message Broker

Some ways to make best use of the tools

ON DEMAND BUSINESS

Agenda

- Introduction
  - Some terms
- Development framework
  - Establishing a team environment
  - Standards
- Coding
  - Initial design
  - Use of broker features
  - Using ESQL
  - Using Java
- Deployment
  - Scripting operations

Who am I

- Andy Piper
  - WebSphere Solutions Specialist working in Software Services
  - co-author of the Redbook *Migration to WBI MB v5* and SupportPac *WBIMB Change Management and Naming Standards*
  - co-author of education on WebSphere Message Broker
  - engagements have included architecture, design, implementation, troubleshooting (... etc)

Development framework

- WMB has several features which improve team working practices
  - We ought to consider processes to work with these features
- It is also important to consider common standards and procedures

Working as a team

There are certain common prerequisites to effective team working

a) Strong communication
b) Shared code (source control)
c) Common standards

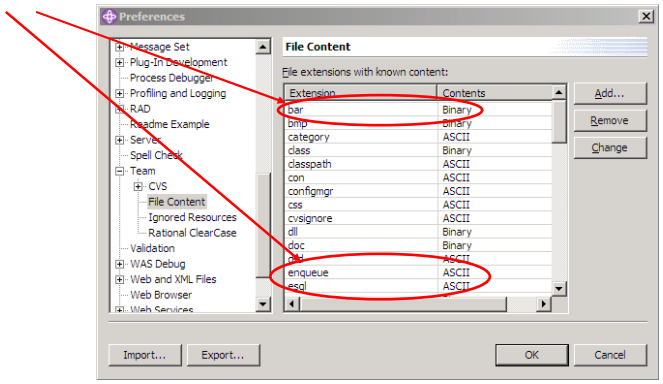
Message Broker can't help us to communicate better as a team… but the tools help us to share code, which should encourage us to pay attention to standards

Source control

- In WMQI v2.1, the CM did check in/out plus resource locking
- WebSphere Message Broker uses the Eclipse architecture
  - Supports CVS out-of-the-box, ClearCase with plugin – see SupportPac IC04, Redbooks
  - Others available via www.eclipse.org (Subversion, VSS, Harvest)
- Establish version control regime early
  - Avoid divergence between developers and projects
- Consider project size when choosing tools
  - Is there a corporate standard?
  - Are the developers geographically dispersed?
  - Are atomic commits important?

Tips for using source control

- Update workbench preferences to understand binary / ASCII files

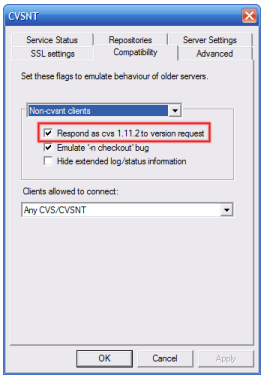


- Remember that many files are XML-based and therefore not easy to diff (.esql and .java files are exceptions)

Further tips for using source control

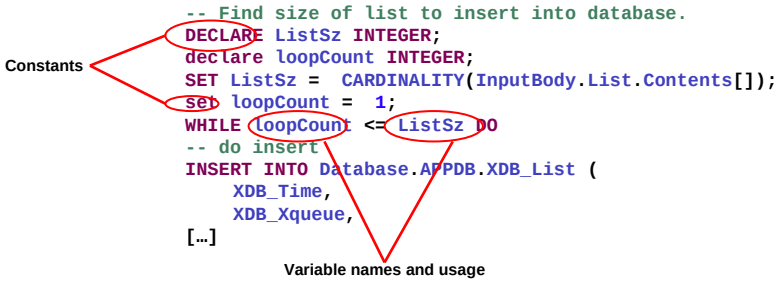
- Include the .project file for Message Flow and Message Set projects
  - otherwise the broker tooling may now work properly… can't add flows to broker archive files.

- Be careful if using CVSNT – certain versions are incompatible with certain levels of Eclipse without workarounds which I'm helpfully pointing out here! →



Standards

- Consistency is important
  - Establishes a “common language” between developers
  - Makes review easier
    NB “Review Code” is a step in the Rational Unified Process



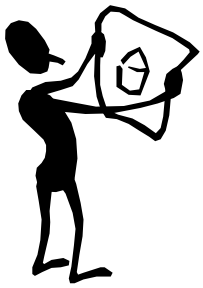
  - Not limited to ESQL and Java – think about readability of flows, node naming

Coding using WebSphere Message Broker

- Message broker is like many other application development tool and runtimes
  - often, several ways to achieve functional requirements
  - ESQL and Java both supported
    - Java now a first-class language alongside ESQL
- It is worth following some common practices

Initial design

- We need to think about how best to implement the functional requirements
- In particular
  - Performance
  - Scalability
  - Backend integration specifics
  - Error handling



Message design

- Well defined header
  - Maximise the possibilities of standard components
  - Reduce complexity and multiple parsing
- Persistent vs. non persistent
  - Use non-persistent where possible
  - Persistent incurs overhead but is best bet for assured once-only delivery
- General performance
  - Format, size, complexity
  - Flow critical path length
  - Use the headers MQMD, MQRFH2 (careful of platform support, improved now bridges ignore RFH2)

Flow design: Performance

- Specific flows
  - Optimised processing
  - Multiple flows to code and manage
  - More process slot taken, maintenance
- Generic flows
  - Fewer flows to manage
  - Processing not optimised
- Many Short flows
  - Short units of work
  - Incur multiple parses
  - Loss of context
  - Higher WMQ cost
- Fewer Long flows
  - Minimise WMQ/disk overhead
  - Flow critical path length

Flow design: Reuse

- Same comments apply re: specific / generic flows
- If multiple protocols are being (or may be) used, worth encapsulating main processing within subflows
  - can be an effective way of providing multiple input channels (HTTP, MQ, etc.) into a single set of function

Remember…

- Flows are *threads*, triggered by message events
  - Message events can be triggered by CRON sending messages, timer events, or a scheduler etc.
  - Brokers are essentially stateless engines.
    - **Keep them stateless** when possible - WMQ clustering gives you excellent linear scalability
    - Aim for **share nothing** designs for the broker topology
    - Place broker databases local to the broker, not on separate DB hosts
    - Tune for either throughput or response time, and separate the workloads
  - Clusters and zero affinity messages allow best ROI for the hardware when used in active/active HA configurations

IBM Software Group

Best practice use of features

- Transformations
- ESQL
- Java
- Flow structure

Transformations

- "Classic": use ESQL
  - `SET OutputRoot.XML.Message.Body = InputBody.A;`
- Java**
- Mappings node
  - specialised Compute node, drag/drop approach
- XSLT
  - long-requested, standards-based, but remember…
  - static, local location of stylesheets (no online capability)
- Custom node (C, Java)
  - always an option, but think about portability, and maintenance,
- Consider the best way to achieve rapid results with good reuse – will depend on your environment, team, skills, etc.

Message Flow ESQL

- Each node has a distinctly named Module
- ESQL Modules created in one file per flow
- Statement execution begins in function Main()
- CALL statements used to split code into separate functions and procedures
- ESQL Functions can take input parameters
- ESQL Functions cannot alter value of input parameters

MessageFlow.esql

```
CREATE COMPUTE MODULE MessageFlow_Compute
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN

        -- CALL CopyMessageHeaders();
        CALL CopyEntireMessage();

        RETURN TRUE;
    END;

    CREATE PROCEDURE CopyMessageHeaders() BEGIN
        DECLARE I INTEGER 1;
        DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
        WHILE I < J DO
            SET OutputRoot.*[I] = InputRoot.*[I];
            SET I = I + 1;
        END WHILE;
    END;

    CREATE PROCEDURE CopyEntireMessage() BEGIN
        SET OutputRoot = InputRoot;
    END;
END MODULE;
```

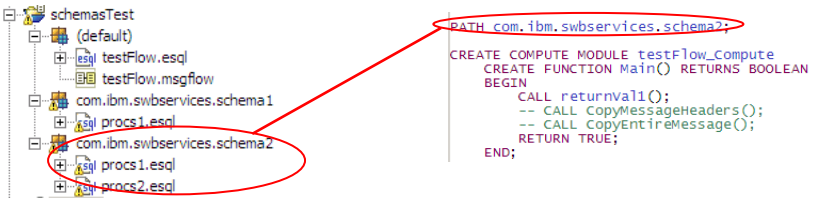
Modularisation of ESQL Code

Build code libraries!

- ESQL “libraries” can contain Functions and Procedures
- PATH statements cannot occur within a Module: only at schema scope
- PATH statements used to search for functions and procedures stored in other broker schemas (like Java `import`)

Broker schemas at deployment and runtime

- ESQL is not deployed in a modular way – functions and procedures are serialised into the flow at deployment time
- This results in some specific behaviour…



- Everything in schema2 will be pulled into the .cmf file at BAR creation time

ESQL enhancements – 1 (of 2)

- Improved support for creating DATETIME variables
  - e.g. `DECLARE myDate DATE = DATE(2004, 03, 01);`
- Flexible type formatting
  - Add FORMAT clause to CAST

```
e.g. SET myString = CAST(money AS CHAR FORMAT
"£#,##0.00");
```

- Cardinality functions
  - Remove existing restrictions on CARDINALITY and EXISTS
  - New SINGULAR function
- Multiple database support
  - Compute, Database, Filter nodes can now access multiple databases

ESQL enhancements – 2 (of 2)

- Broker Attributes
  - Allows flow designer to determine state

```
DECLARE myFlow CHARACTER = MessageFlowName;
DECLARE myProcess CHARACTER = ProcessId;
DECLARE brokerPlatform CHARACTER = Family;   -- “UNIX”;
DECLARE brokerQueueManager CHARACTER = QueueManagerName;
```

- User Defined Attributes
  - Attributes of a node that can be defined and given values by the user
  - Can be accessed by ESQL of a node (treated as constants)

```
DECLARE COLOUR EXTERNAL CHARACTER;
```

- SQL Handlers
  - Run when error condition is detected
  - Provides more flexible and improved support for handling of errors

Semi-Persistent Environment

- ESQL provides data types with lifetimes longer than current message
- Improved performance in many scenarios
  - Static routing table
  - Counting messages, assigning sequence numbers
- Read and Write usage
  - Read-only usage for database table caching
  - Read-write usage for updates
- Shared across threads for Nodes and Flows
  - Simplifies programming model
- Semi-persistent
  - Maintained between messages, but NOT maintained over execution group restart – “semi persistent”

Intelligent ESQL Coding Practices “DO”s

- Initialise variables within DECLARE statements
- Consider setting multiple variables of the same type in single DECLARE
- Declare REFERENCEs to avoid excess navigation of Message Tree
- Use CREATE with PARSE clause in preference to a read / write operation
- Make code as concise as possible … restricting the number of statements cuts any parsing overhead
- Use LASTMOVE or CARDINALITY statements to check the existence of fields in the message tree … avoids mistakes

Intelligent ESQL Coding Practices “DON’T”s

- Do not use CARDINALITY statements inside loops
- Avoid overuse of Compute nodes … tree copying is processor heavy (this can also make us think harder about subflows)
- Avoid nested IF statements: use ELSEIF or better still, CASE / WHEN clauses (quicker drop-out)
- String manipulation is processor heavy – do not overuse
- (use REPLACE function in preference to a complete re-parsing)

Tip:

The practices described here are covered in more detail in various developerWorks articles by Tim Dunn and Kevin Braithwaite – well worth a read if you are looking for performance advice

Java… how and when to use it

- We now have multiple ways to use Java
  - Plugin nodes
  - Java Compute node **
  - Calling static methods from ESQL (since WBIMB v5 Fixpack 4, using `CREATE PROCEDURE ... LANGUAGE JAVA`)
  - Calling a web service over e.g. HTTP
- Consider
  - Transactionality
  - Maintenance
  - Performance
  - Deployment issues

Java as a First Class Transformation Language

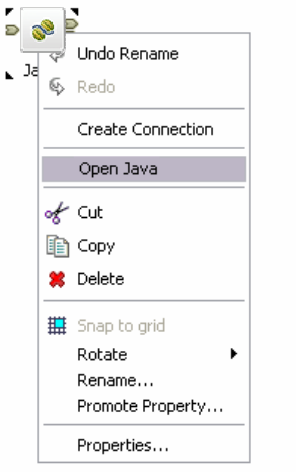
- General purpose programmable node
  - Java programming language
  - High Performance for processing logic and tree access
- Offers “Compute node” alternative for Java programmers
  - Similar “look and feel”
  - No ESQL skill or experience required
- Extra convenience methods have been added
  - The message tree can be queried and traversed using XPath 1.0 syntax
  - Extensions to allow new elements to be created in message structure
- Databases can be accessed via two supported routes
  - JDBC type 4 drivers - standard Java, non-transactional
  - MbSQLStatement - uses broker’s ESQL syntax, fully transactional

Using the Java Compute Node

- Full Eclipse Java experience
  - Eclipse Java editor provides built-in syntax assists
- Select which template to use:
  - Read-only messages
  - Transforming messages (read/write)
- Java class is a property of the node
  - Equivalent to the ESQL module in a Compute node

```
public class jcn2 extends MbJavaComputeNode {
  public void evaluate(MbMessageAssembly assembly)
              throws   MbException
  {
    MbOutputTerminal out = getOutputTerminal("out");
    MbOutputTerminal alt = getOutputTerminal("alternate");
    MbMessage message = assembly.getMessage();
    // Add user code below

    // End of user code
    out.propagate(assembly);
  }
```



Messaging Processing Nodes: New and updated

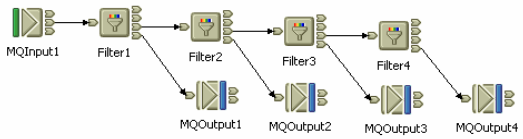
New

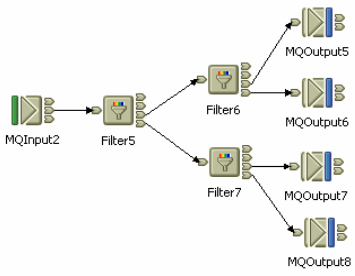
- Java Compute node
  - Provide existing Compute node capability for Java programmers
  - Deploy Java JARs
- TimerControl Node
  - One shot, Periodic, N shot (persistent and non persistent)
- MQGET node
  - Simple aggregation and/or mechanism to hold state
- JMS Input/Output
  - Native JMS Interoperability

Updated

- Web Services
  - HTTPS support
- Aggregation
  - MQ based implementation
  - Delivers improved performance
- XSLT
  - Deployed style sheets
  - Compiled style sheets

Flow structure… Filter nodes

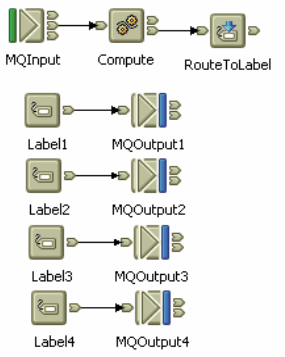




- Intelligent Routing of messages within flows is possible using the RouteToLabel or Filter methods
- If using a Filter method be sensitive to the relative volumes of each message type likely to be passed through the flow.
- Design flows so the majority of messages flow through the minimum number of possible nodes

Flow structure… RouteToLabel nodes

- RouteToLabel method requires a preceding Compute node to set the DestinationList area of the message tree
- RouteToLabel routes to either the first or last entry in DestinationList
- The number of Label nodes is insignificant for performance (message only takes one path)
- The cost for performance is in setting the DestinationList (you may have to set an entry for all Labels in the DestinationList even though the message is only routed down one branch)
- No real cost in reading DestinationList



Scripting operations

- The deployment model is now considerably more flexible
- New features enable many more operations to be scripted
- We can benefit from the flexibility but need to think about how to manage it

Things to consider

- Configuration management techniques apply to the deployment stage just as much as to development
- We should script, and control:
  - Database table definitions
  - Queue manager definitions (mqsc scripts - saveqmgr MS03 – already updated for WMQ v6)
- A ‘release’ = flows, plugin nodes, queue definitions, DB definitions (plus actual data), additional Java classes
- Only the binaries and configurations should be distributed between environments – use source control to look after sources

Scripted deployment, improved administration

- New command line tools
  - Start/Stop message flows
  - Create/Delete execution groups
- Java administration API (“Configuration Manager Proxy”)
- Runtime versioning
- Restart database without restarting the broker

Command Line Administration – New and Improved

- ☑ mqsimigratecomponents
- ☑ mqsimigratemfmaps
- ☑ **mqsicreateexecutiongroup**
- ☑ mqsideleteexecutiongroup
- ☑ **mqsistartmsgflow**
- ☑ **mqsistopmsgflow**
- ☑ **mqsibackupconfigmgr**
- ☑ **mqsirestoreconfigmgr**
- ☑ mqsicreatedb
- ☑ mqsideletedb
- ☑ mqsicreateaclentry
- ☑ mqsideleteaclentry
- ☑ mqsilistaclentry
- ★ Create your own!

New

- ☑ mqsideploy
- ☑ mqsilist
- ☑ mqsicreatebar
- ☑ mqsicreatebroker
- ☑ mqsicreateconfigmgr
- ☑ mqsicreateusernameserver
- ☑ mqsisetdbparms
- ☑ mqsichangebroker
- ☑ mqsichangeconfigmgr
- ☑ mqsichangeusernameserver
- ☑ mqsideleteconfigmgr

Improved

Configuration Manager Proxy API

- A complete Java programming interface to the Configuration Manager
- Administer domains programmatically
  - Brokers
  - Execution groups
  - Message flows
  - Dictionaries
  - Subscriptions
  - Topology
  - Collectives
  - Event Log
  - Topics
  - Configuration Manager

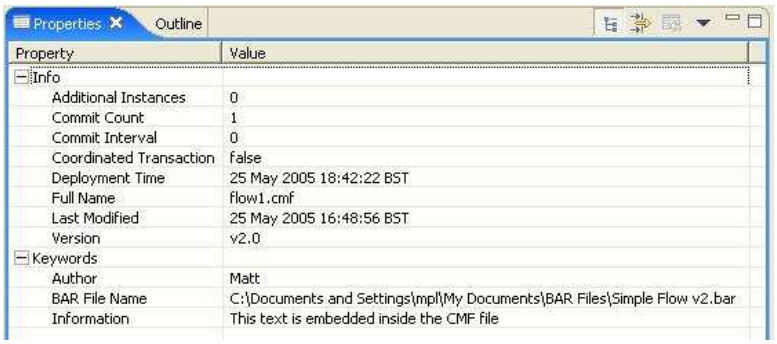
```
import com.ibm.broker.config.proxy.*;

public class CreateBroker {
  public static void main(String[] args) {
    ConfigManagerProxy cmp =
              ConfigManagerProxy.getInstance(…);
TopologyProxy topology = cmp.getTopology();
    topology.createBroker("MYBROKER","QMGR");
  }
}
```

- Comprehensive samples and documentation provided – build your own tools

Runtime versioning

- It is now easier to discover what has been deployed to your brokers
- New fields associated with each deployed object
  - Deployment time, Modification time, BAR file name, Version



Developing an automation framework

A tool such as Ant can be used to assist with Message Broker deployment:

- Check out / tag from source control
- Build broker archives → NB mqsicreatebar = headless Eclipse, Windows and Linux only
- Build JARs for Java libraries and nodes
- FTP / Telnet / SSH to remote hosts (restarts)
- Deploy…

Similar results can be achieved with Windows batch files, or UNIX shell scripts; but Ant is highly extensible

A typical process (1)

- Developers check the code into source control
  - If using CVS, include version information tags in source
- Ant script:
  - Checks out current stream
  - Applies a version tag or label
  - Builds broker archive and JARs as required
    - Including version tags (mqsicreatebar -version)
  - Copies deployment artefacts to central location

A typical process (2)

- Ant script *optionally*:
  - FTPs / SCPs files to remote hosts
  - Telnets / SSHs to remote hosts to:
    - restart broker (if Java nodes included)
    - deploy DDL and mqsc scripts
  - Creates execution groups and deploys flows (NB JARs and XSLTs now carried in the BAR)
- These optional steps are usually fine for test environments, but may need closer control in production

Reading list

- A very accessible and readable book on Ant
  - *Java Development with Ant*, Erik Hatcher and Steve Loughran
- Excellent articles on all aspects of WMB design and performance
  - www.ibm.com/developerworks
    (search terms = “broker”, “WBIMB”, “Dunn”)

In conclusion…

- Know your tools and materials
  - Understand the technologies you are using
- Be consistent
  - when coding, when deploying
- Consider the options when coding – choose the best methods to achieve functional and performance requirements
- Follow good team working and coding practices
- Be prepared to set aside some initial time to get scripted deployment operations right

IBM Software Group

Questions

Thank You

Andy Piper

andy.piper@uk.ibm.com

ON DEMAND BUSINESS