Under the hood of Message Broker on z/OS (WLM, SMF and More)

David Gorman
(gormand@uk.ibm.com)
IBM

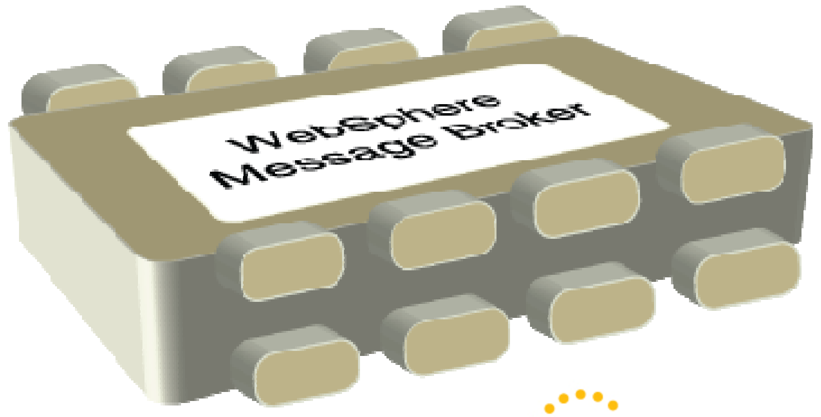
10th August 2011
Session Number (09437)

Agenda

- Message Broker Overview.
  - What is Message Broker
  - Components of Message Broker
  - Message Flows
  - Patterns
  - Message Broker Toolkit
  - Message Broker Explorer
- Message Broker on z/OS.
- Value of Message Broker on z/OS.

What is WebSphere Message Broker?

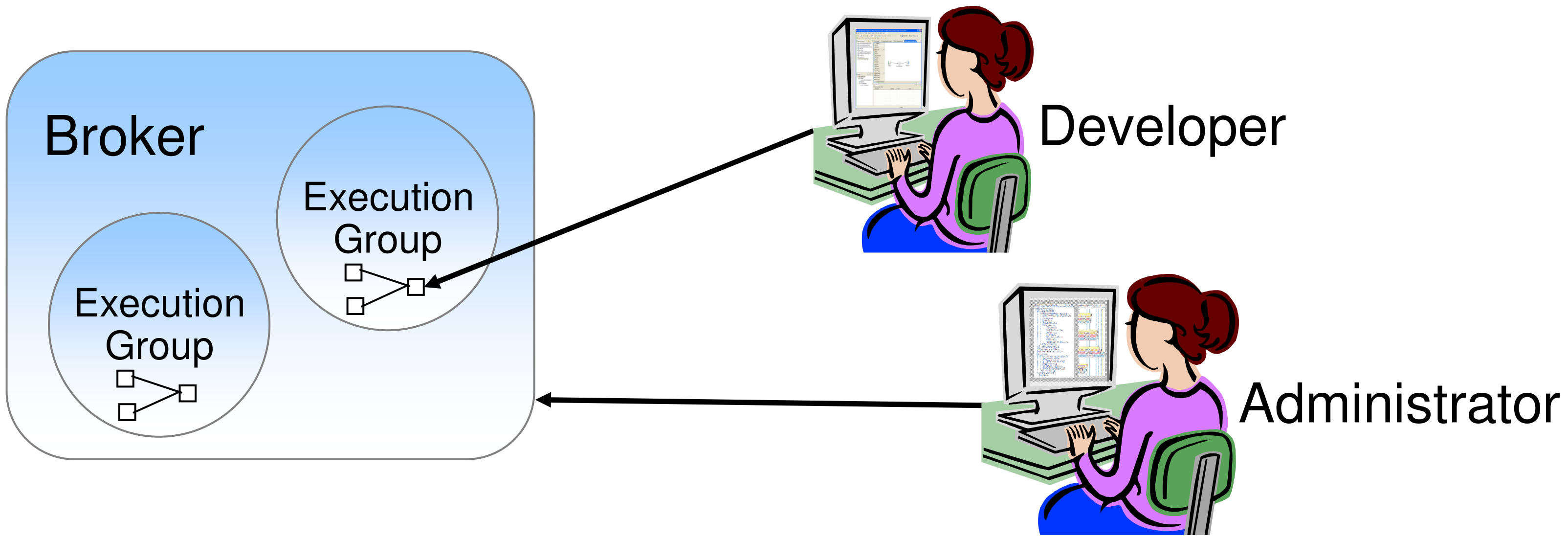
- Message Broker enables “universal connectivity” by integrating protocols, message formats and mediation patterns
  - Emphasis on application re-use
- Fits naturally with WebSphere MQ
  - Robust, scalable architecture
  - Optimized for high throughput
  - Flexible broker topologies
- Three programming constructs are used:
  - Message Flows
  - Nodes
  - Message Models



Notes: What is WebSphere Message Broker?

- **Universal Connectivity**
  - Simplify application connectivity to provide a flexible and dynamic infrastructure
- **Routes and transforms messages FROM anywhere, TO anywhere**
  - Supports a wide range of transports, protocols & systems
    - *MQ, JMS 1.1, HTTP(S), SOAP, REST, File (incl. FTP & FTE), Database, TCP/IP, MQTT…*
    - *CICS, IMS, SAP, SEBL, PeopleSoft, JDEdwards, SCA, CORBA, email…*
  - Supports a broad range of data formats
    - *Binary (C/COBOL), XML, CSV, Industry (SWIFT, EDI, HL7…), IDOCs, User Defined*
  - Message Processors
    - *Route, Filter, Transform, Enrich, Monitor, Distribute, Decompose, Sequence, Correlate, Detect*
- **Simple programming**
  - Patterns based for top-down, parameterized connectivity of common use cases
    - *e.g. Web Service façades, Message oriented processing, Queue to File…*
  - Construction based for bottom-up assembly of bespoke connectivity logic
    - *Message Flows to describe application connectivity comprising…*
    - *Message Nodes which encapsulate required integration logic which operate on…*
    - *Message Tree which describes the data in a format independent manner*
    - *Transformation options include Graphical mapping, PHP, Java, ESQL, XSL and WTX*
- **Operational Management and Performance**
  - Extensive Administration and Systems Management facilities for developed solutions
  - Wide range of operating system and hardware platforms supported, including virtual & WCA Hypervisor
  - Offers performance of traditional transaction processing environments
  - Deployment options include Trial, Remote Deployment, GetStarted, Enterprise

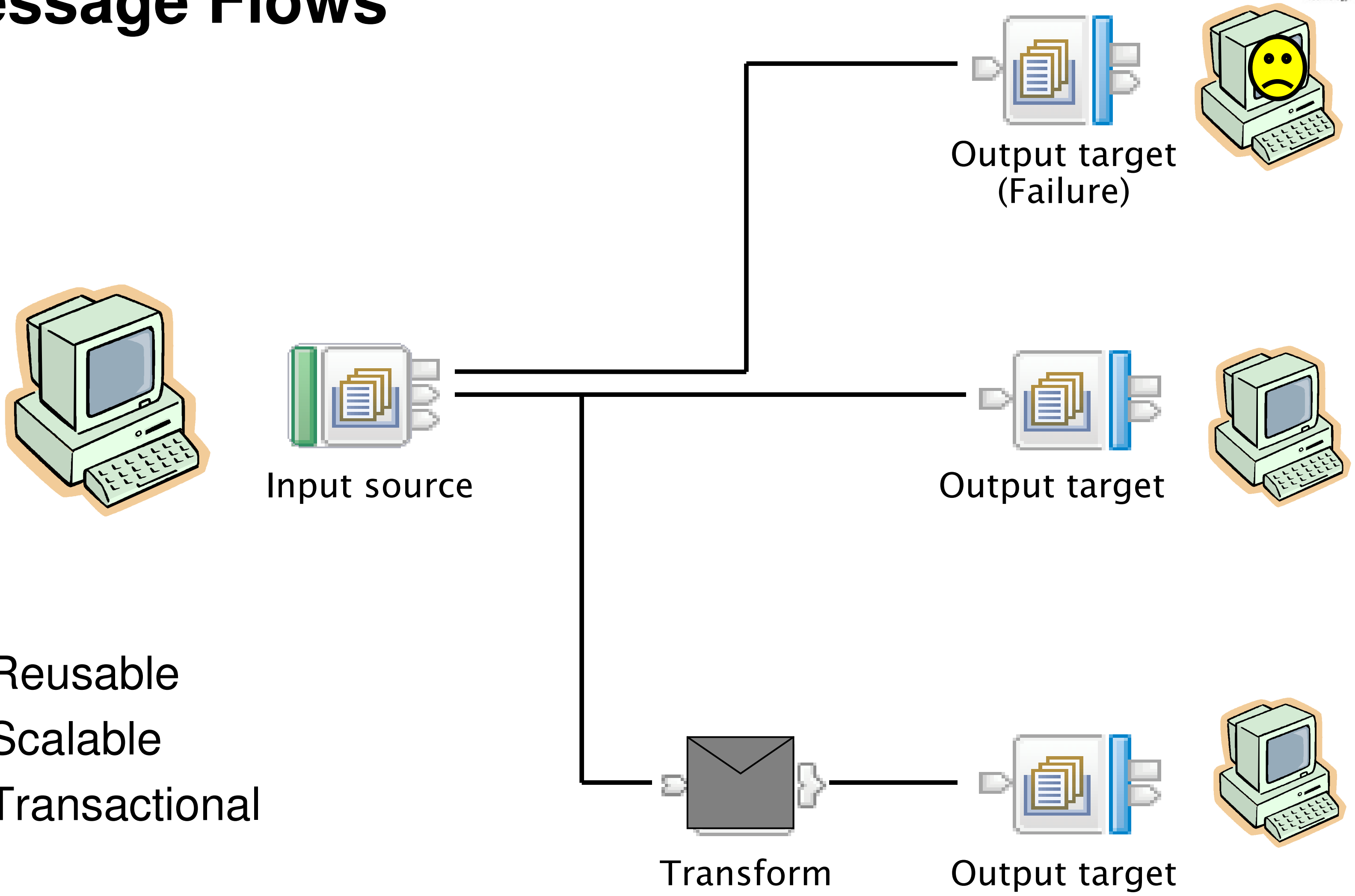
Components of Message Broker



- Developer
  - Development and Test Environment called Message Broker toolkit
  - Built on Rational Application Developer
- Administrator
  - Advanced Administration Tool called Message Broker Explorer
  - Built on MQ Explorer
  - Java CMP API for scripting
  - Commands

- Broker
  - Standalone runtime environment that runs message flows
  - Execution groups for isolation and scalability
  - Flows contain connectivity and transformation logic
  - Many different platforms
  - Builds on an MQ queue manager

Message Flows



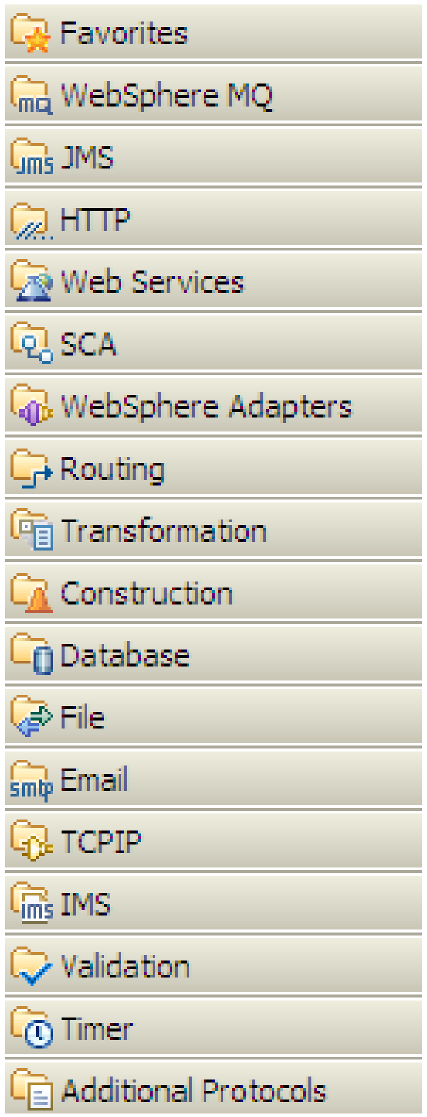
- Reusable
- Scalable
- Transactional

Notes : Message Flows

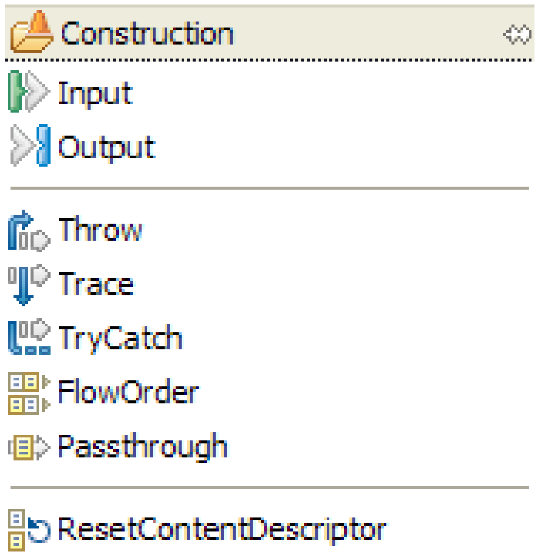
- Message flows provide the processing sequence required to connect applications together.
- A message flow contains the set of operations required to take a message from an originating application and deliver copies of it, some possibly transformed, to any number of connected applications for processing.
- As a message passes through a message flow, it is transformed and routed according to the nodes it encounters, and the processing decisions made within those nodes. Later we'll see how the nodes can modify the values within, or transform the structure of, a message to provide the data transformations necessary to drive backend server applications.
- For a given application scenario, the message flow describes all possible outcomes when processing a message. For example, if the message has a high monetary value, a copy of it might have to be routed to an audit application. Or if the message is not well-formed (may be it's not encrypted in the right format), it might be routed to a security application to raise an alert.
- Equally important is the visualization of the application integration within then organization. Very often, for any particular application scenario, the application connectivity requirements (*business*!) is held within the heads of domain experts. Being able to view the integration structure brings benefits in scenario understanding, reuse potential, and application architecture/standards conformance.
- After a message has been processed by a message flow, the flow does not maintain any state. It is possible to maintain such state in an external database, or within the message by using an extensible header such as the MQRFH2.
- Message flows are transactional.
  - Message flows provide vital processing and data manipulation and are therefore fully transactional. A message flow either completes all or none of its processing successfully.
  - However, if required, individual nodes can elect to perform operations outside of the message flow transaction. (e.g. audit)
- Message flows are multithreaded.
  - A given message passing through a series of nodes will execute on a single thread. To allow increased message throughput, message flows can be defined with many additional threads assigned to them. Peak workloads use additional threads, which are pooled during inactivity. We'll see more implementation details later. This means application scaling can be an operational rather than design time decision.
- Message flow nesting and chaining allow construction of enhanced capabilities.
  - Sophisticated flows can be rapidly constructed by linking individual flows together as well as nesting flows within each other.
- References:
  - Message Flow overview at http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac00310_.htm

Notes : Nodes

- The building blocks of message flows
- Each node type performs a different (input, output or processing) action
- Many different node types
  - Grouped into logical categories in the message flow editor


Favorites
WebSphere MQ
JMS
HTTP
Web Services
SCA
WebSphere Adapters
Routing
Transformation
Construction
Database
File
Email
TCPIP
IMS
Validation
Timer
Additional Protocols

Notes : Built-in Nodes for Routing, Transformation and Event processing



Construction

Input

Output

Throw

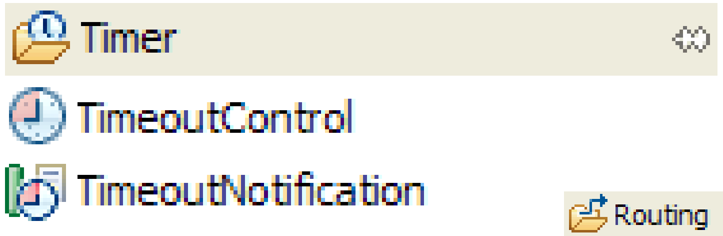
Trace

TryCatch

FlowOrder

Passthrough

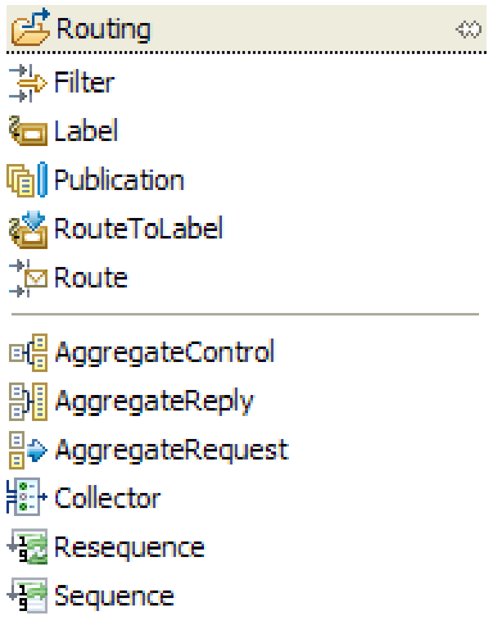
ResetContentDescriptor



Timer

TimeoutControl

TimeoutNotification



Routing

Filter

Label

Publication

RouteToLabel

Route

AggregateControl

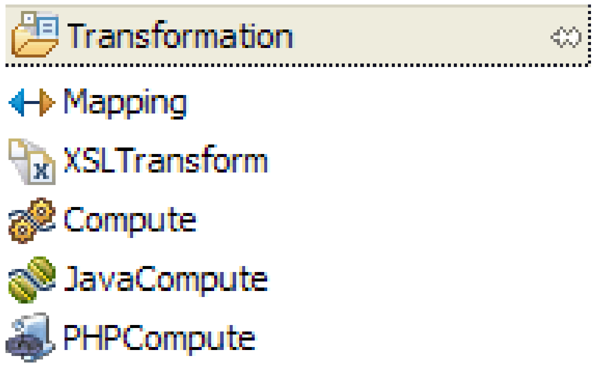
AggregateReply

AggregateRequest

Collector

Resequence

Sequence



Transformation

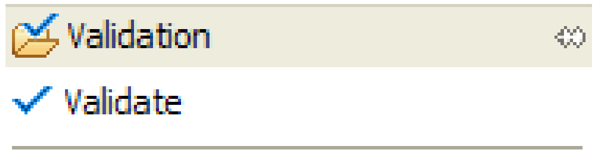
Mapping

XSLTransform

Compute

JavaCompute

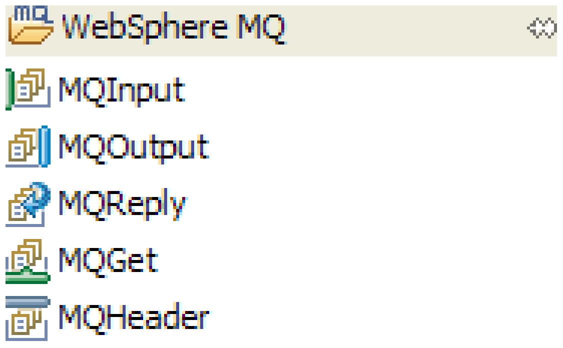
PHPCompute



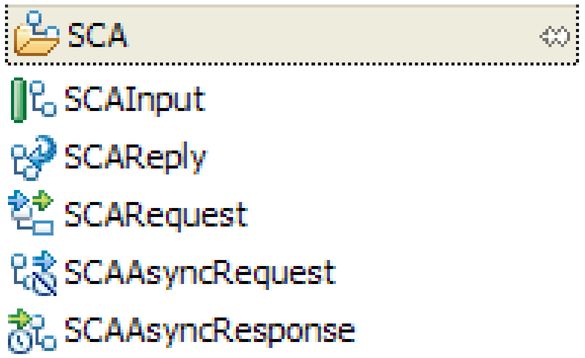
Validation

Validate

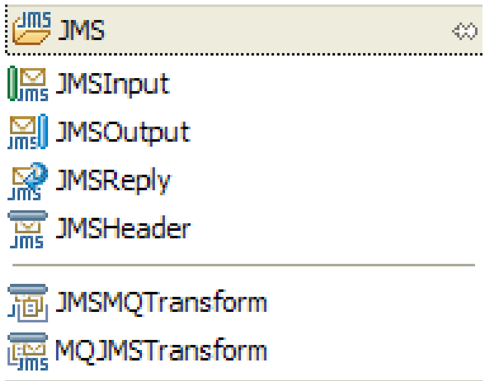
Notes : Built-in Nodes for Transport


WebSphere MQ

- MQInput
- MQOutput
- MQReply
- MQGet
- MQHeader

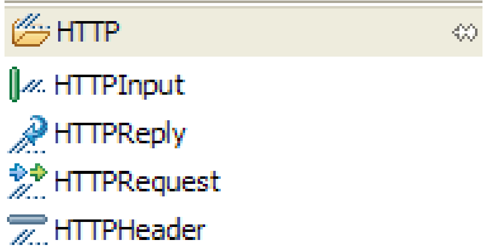

SCA

- SCAInput
- SCAReply
- SCARequest
- SCAAsyncRequest
- SCAAsyncResponse

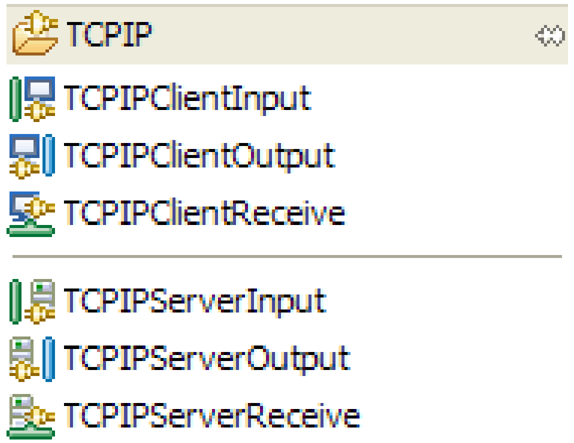

JMS

- JMSInput
- JMSOutput
- JMSReply
- JMSHeader

- JMSMQTransform
- MQJMSTransform

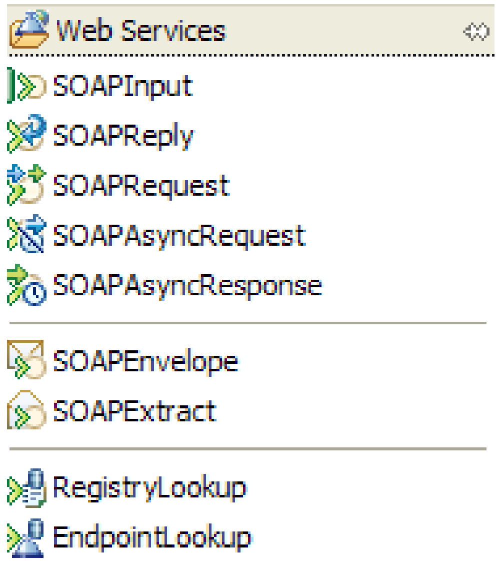

HTTP

- HTTPInput
- HTTPReply
- HTTPRequest
- HTTPHeader


TCPIP

- TCPIPClientInput
- TCPIPClientOutput
- TCPIPClientReceive

- TCPIPServerInput
- TCPIPServerOutput
- TCPIPServerReceive

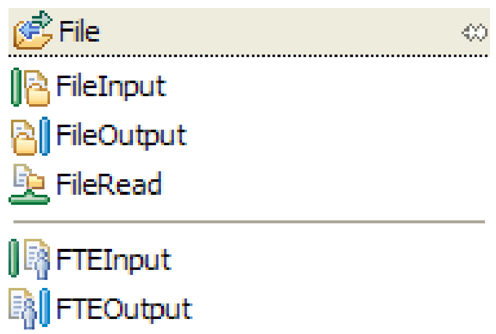

Web Services

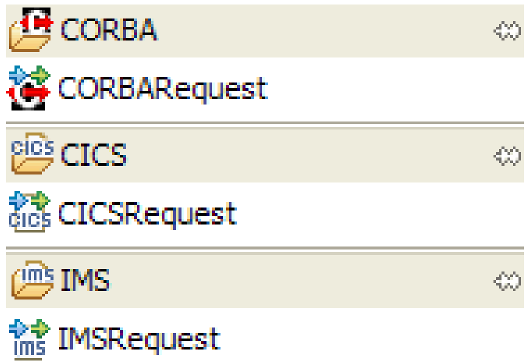
- SOAPInput
- SOAPReply
- SOAPRequest
- SOAPAsyncRequest
- SOAPAsyncResponse

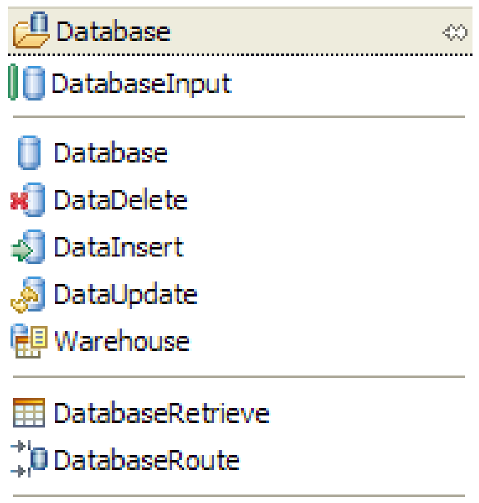
- SOAPEnvelope
- SOAPExtract

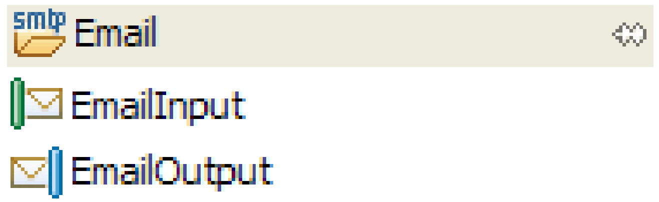
- RegistryLookup
- EndpointLookup

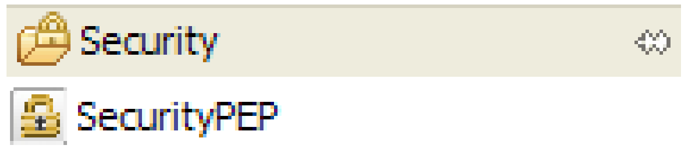
Notes : Built-in Nodes for External Resources

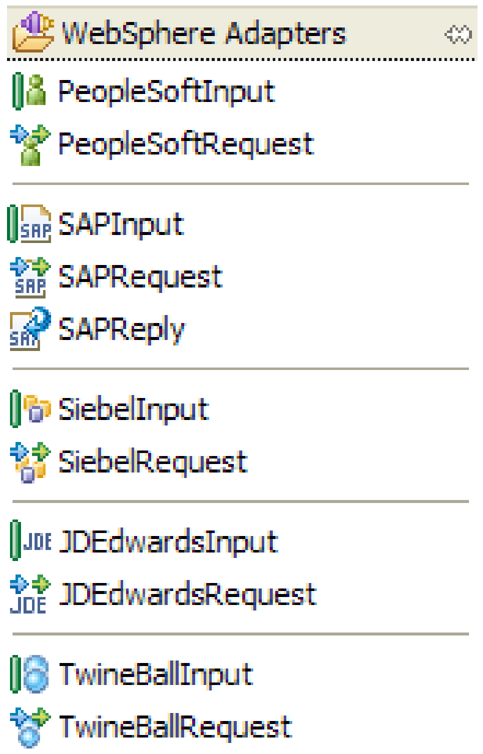

File
FileInput
FileOutput
FileRead
FTEInput
FTEOutput


CORBA
CORBARequest
CICS
CICSRequest
IMS
IMSRequest

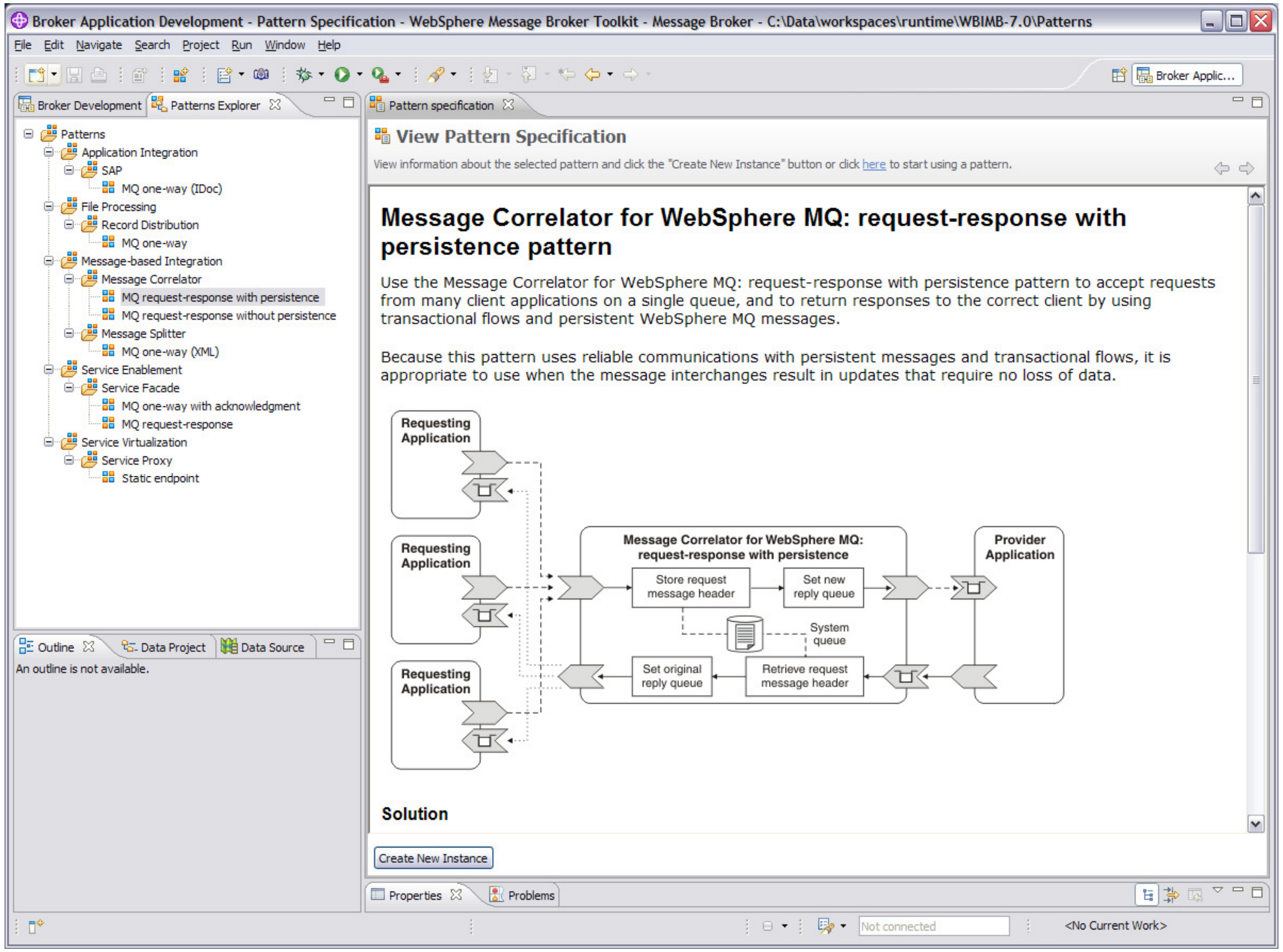

Database
DatabaseInput
Database
DataDelete
DataInsert
DataUpdate
Warehouse
DatabaseRetrieve
DatabaseRoute


Email
EmailInput
EmailOutput


Security
SecurityPEP


WebSphere Adapters
PeopleSoftInput
PeopleSoftRequest
SAPInput
SAPRequest
SAPReply
SiebelInput
SiebelRequest
JDEdwardsInput
JDEdwardsRequest
TwineBallInput
TwineBallRequest

Patterns



Broker Application Development - Pattern Specification - WebSphere Message Broker Toolkit - Message Broker - C:\Data\workspaces\runtime\WBIMB-7.0\Patterns
File Edit Navigate Search Project Run Window Help
Broker Applic...
Broker Development
Patterns Explorer
Patterns
Application Integration
SAP
MQ one-way (IDoc)
File Processing
Record Distribution
MQ one-way
Message-based Integration
Message Correlator
MQ request-response with persistence
MQ request-response without persistence
Message Splitter
MQ one-way (XML)
Service Enablement
Service Facade
MQ one-way with acknowledgment
MQ request-response
Service Virtualization
Service Proxy
Static endpoint
Outline
Data Project
Data Source
An outline is not available.
Pattern specification
View Pattern Specification
View information about the selected pattern and click the "Create New Instance" button or click here to start using a pattern.
Requesting Application
Requesting Application
Requesting Application
Message Correlator for WebSphere MQ: request-response with persistence
Store request message header
Set new reply queue
System queue
Set original reply queue
Retrieve request message header
Provider Application
Create New Instance
Properties
Problems
Not connected
<No Current Work>

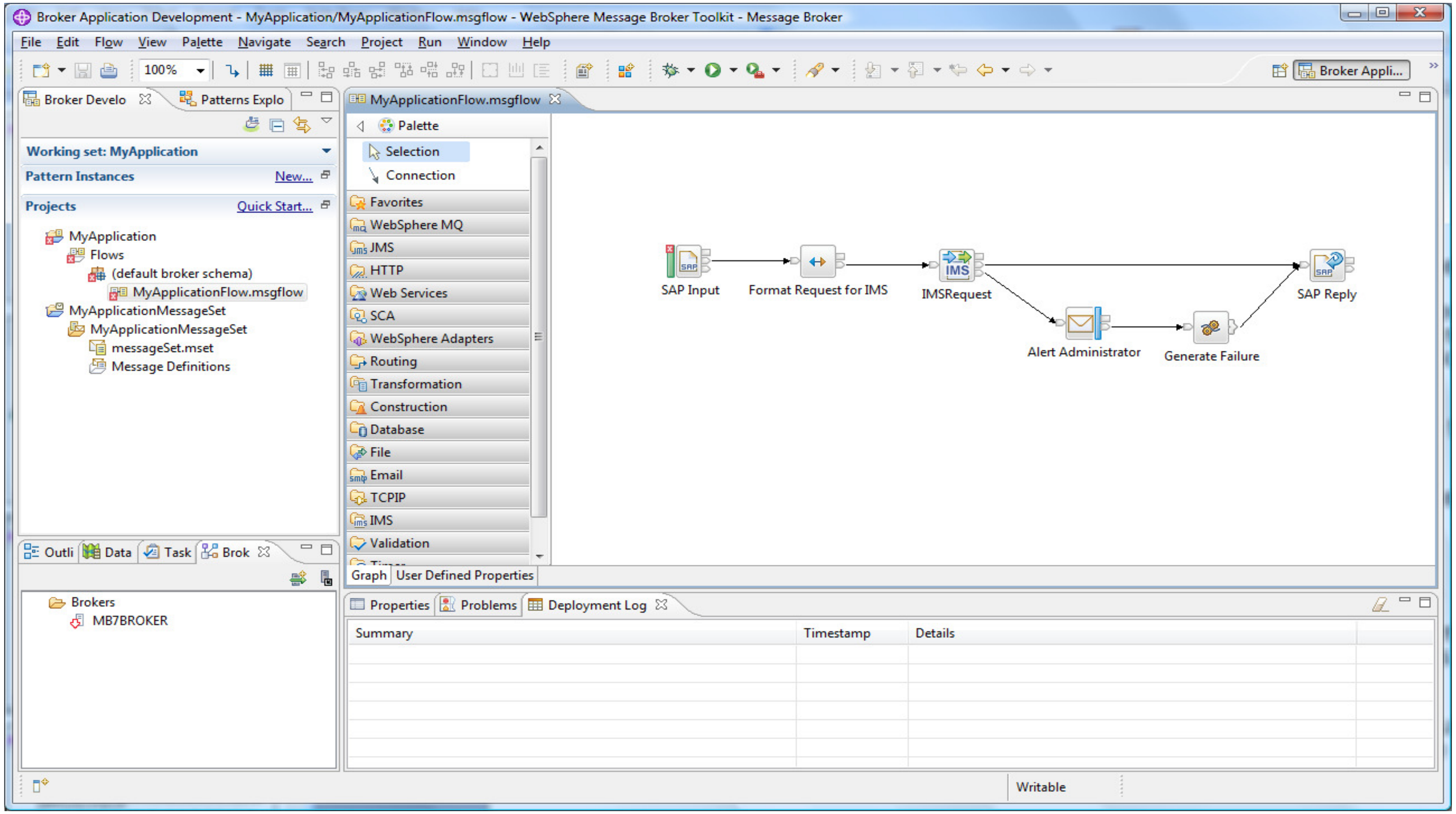
Message Correlator for WebSphere MQ: request-response with persistence pattern

Use the Message Correlator for WebSphere MQ: request-response with persistence pattern to accept requests from many client applications on a single queue, and to return responses to the correct client by using transactional flows and persistent WebSphere MQ messages.

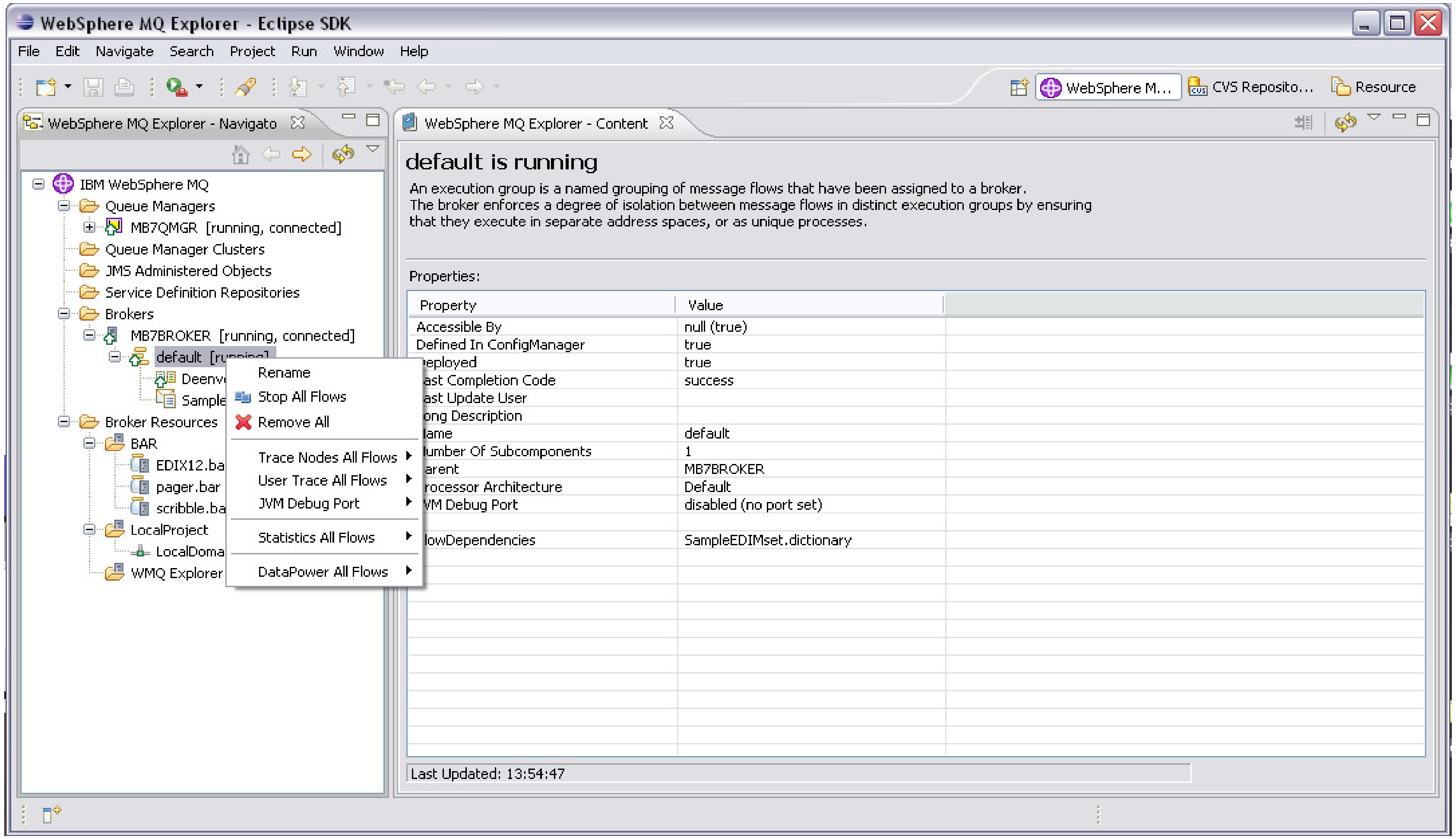
Because this pattern uses reliable communications with persistent messages and transactional flows, it is appropriate to use when the message interchanges result in updates that require no loss of data.

Solution

Application Development Message Flow Editor



Administration
Message Broker Explorer



default is running

An execution group is a named grouping of message flows that have been assigned to a broker.
The broker enforces a degree of isolation between message flows in distinct execution groups by ensuring that they execute in separate address spaces, or as unique processes.

Properties:

Property	Value
Accessible By	null (true)
Defined In ConfigManager	true
eployed	true
ast Completion Code	success
ast Update User	
ong Description	
ame	default
umber Of Subcomponents	1
arent	MB7BROKER
rocessor Architecture	Default
VM Debug Port	disabled (no port set)
lowDependencies	SampleEDIMset.dictionary

Last Updated: 13:54:47

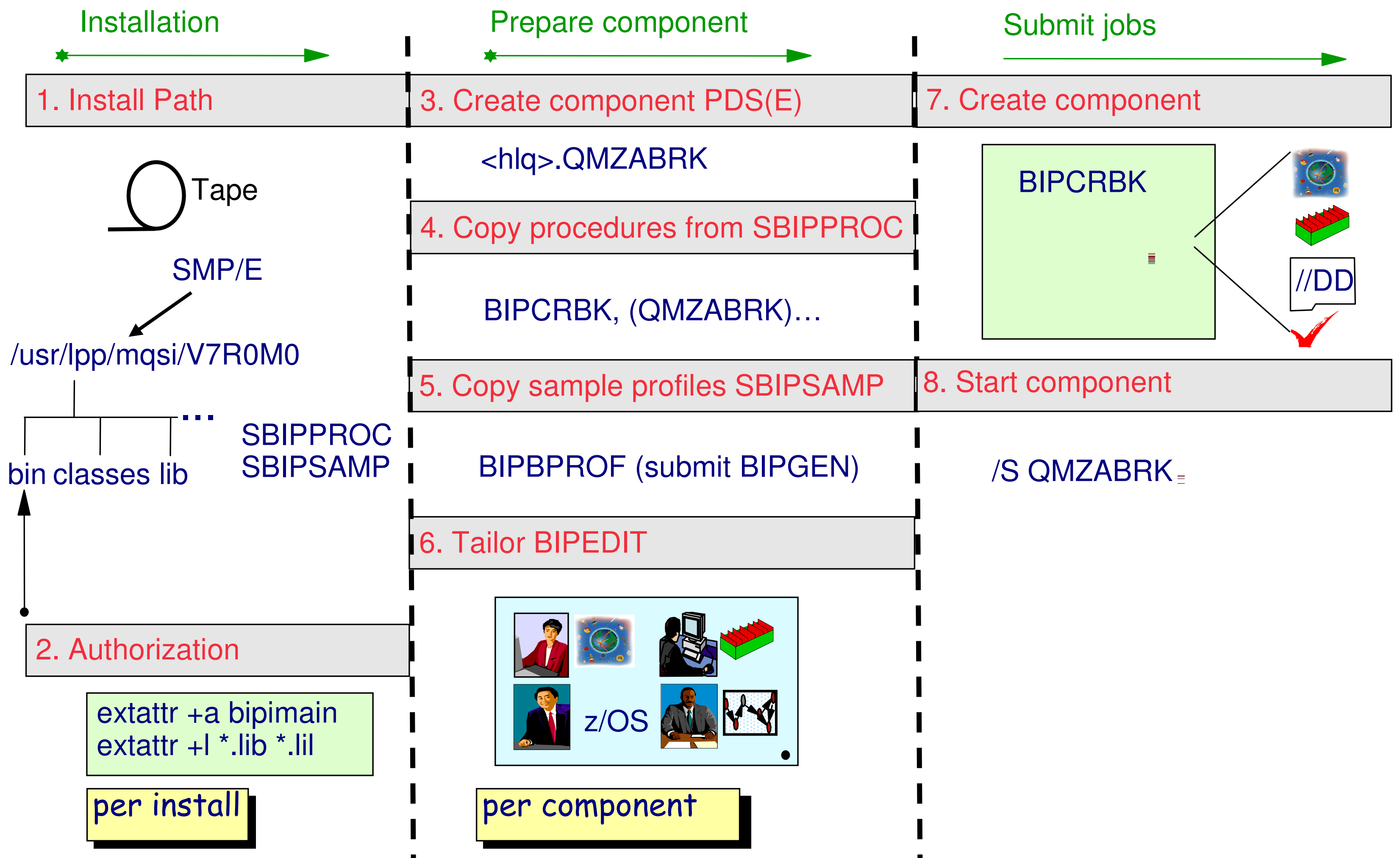
Agenda

- Message Broker Overview
- Message Broker on z/OS.
    - V7 Highlights
    - Install and Customization
    - Starting the broker
    - Displaying broker output
    - Database Connectivity
    - Transaction Model
- Value of Message Broker on z/OS.

V7 Highlights

- Simplification
  - Broker system database removal
    - Only MQ and Java pre-reqs
  - Removal of the Configuration Manager and UserNameServer components
- Enhancements
  - 64bit, Broker is now a true 64bit application
  - Execution Group specific profiles
  - WLM, Assign service classifications to individual address spaces.
  - IBM Sterling ConnectDirect nodes (IC75621)

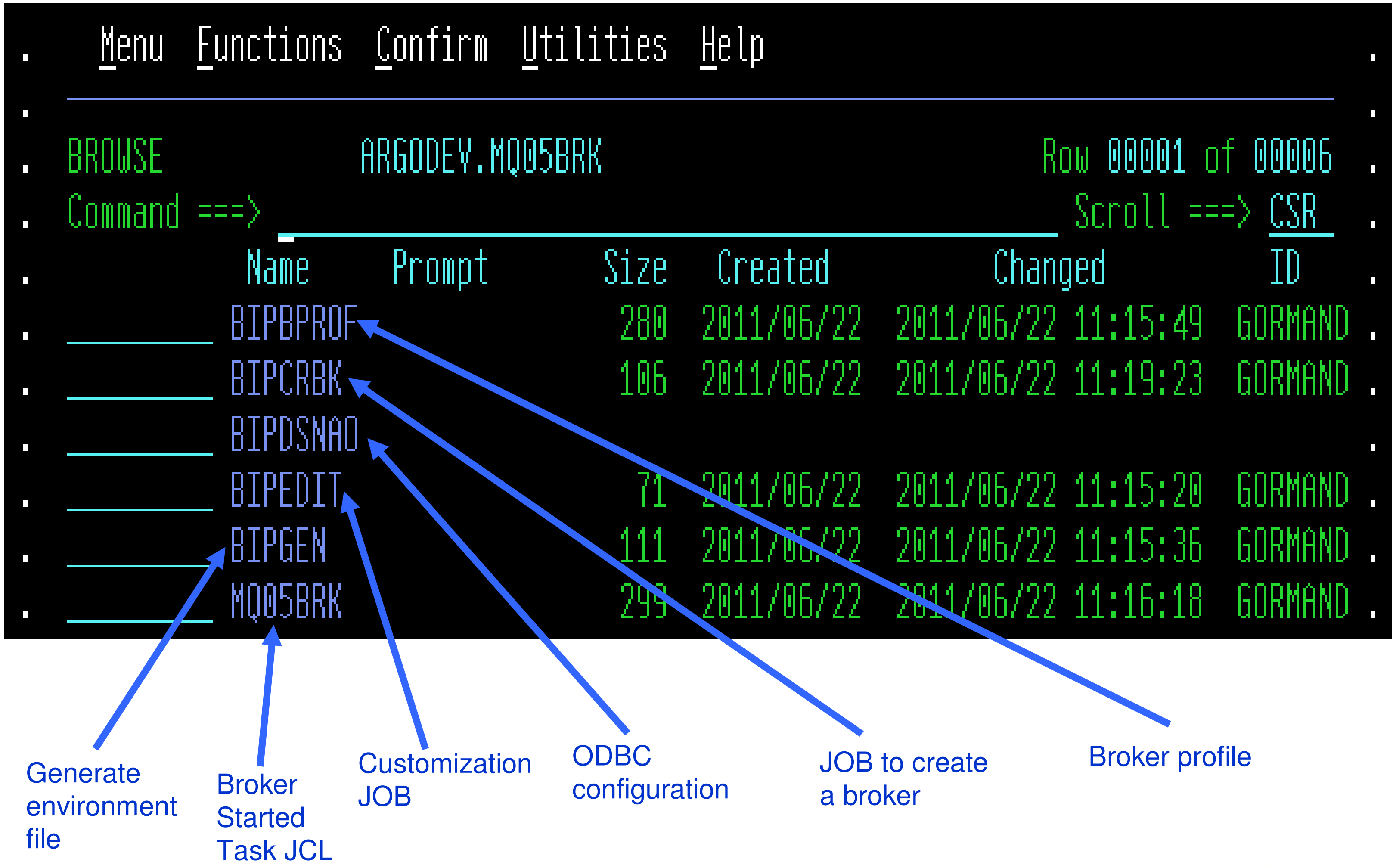
Installation and Customization



Notes : Installation and Customization

- **Install Message Broker via SMP/e.**
- **Allocate Broker PDS(E)**
  - Copy required component procedures from SBIPPROC
  - Rename component as required e.g. QMZAxxxx
- **Customize component dataset**
  - Copy profiles from SBIPSAMP
    - BIPBPROF - Broker profile
  - Other parameter files
    - BIPEDIT – JCL edit file
- **Tailor BIPEDIT JCL edit procedure**
  - Assign settings for registry, MQ, Java etc
- **Target appropriate profile with BIPEDIT**
  - Use BIPGEN to create ENVFILE
- **Target appropriate jobs with BIPEDIT**
  - BIPCRBK– Create Broker
  - Optionally modify options according to authority
    - Default is all resources
      - *-1 only create the registry in USS. (USS Administrator).*
      - *-2 only do the MQ creation pass (MQ Administrator).*
- **Create component**
  - Submit BIPCRBK to create component
- **Target component JCL with BIPEDIT and rename**
  - e.g. BIPBRKP
- **Start component**
  - e.g. /S QMZABRK
- **Component verification automatically runs as the first step**

JCL



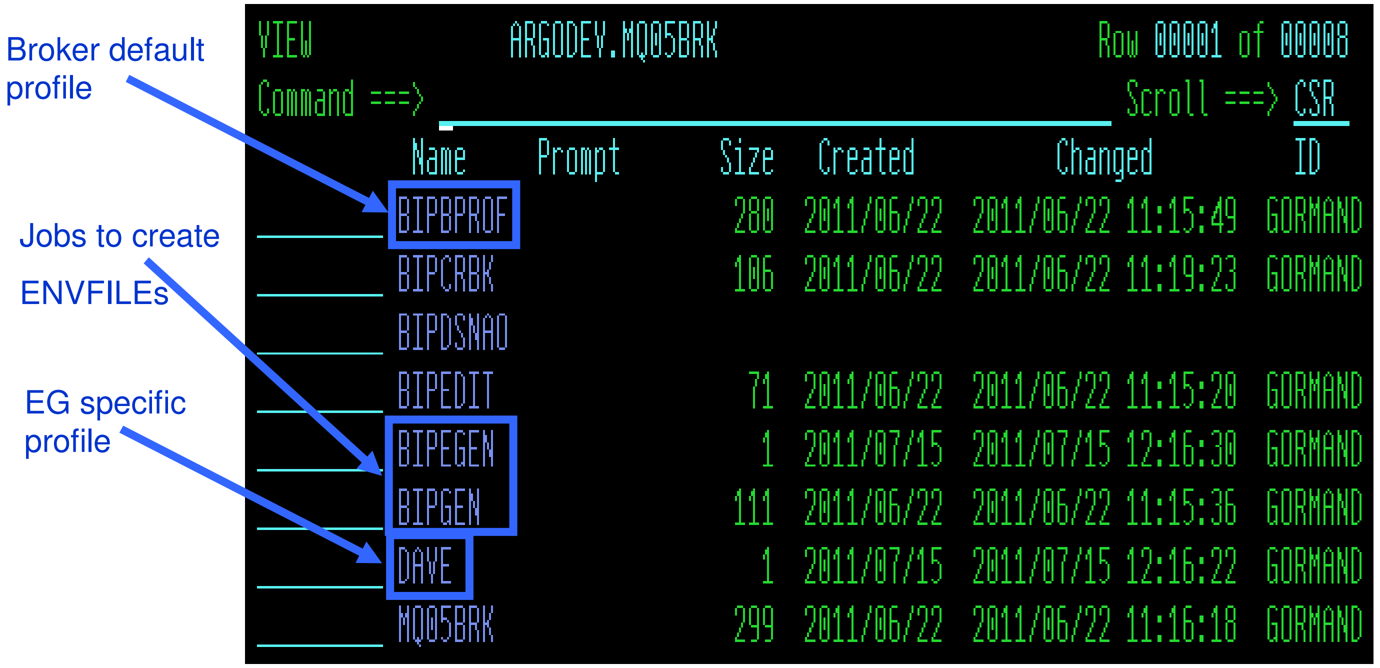
```
  Menu  Functions  Confirm  Utilities  Help
 ------------------------------------------------------------------------------
 BROWSE           ARGODEV.MQ05BRK                           Row 00001 of 00006
 Command ===>                                                  Scroll ===> CSR
            Name     Prompt       Size  Created     Changed              ID
 _________ BIPBPROF                280  2011/06/22  2011/06/22 11:15:49  GORMAND
 _________ BIPCRBK                 106  2011/06/22  2011/06/22 11:19:23  GORMAND
 _________ BIPDSNAO
 _________ BIPEDIT                  71  2011/06/22  2011/06/22 11:15:20  GORMAND
 _________ BIPGEN                  111  2011/06/22  2011/06/22 11:15:36  GORMAND
 _________ MQ05BRK                 299  2011/06/22  2011/06/22 11:16:18  GORMAND
```

A full set of JCL exists for all Message Broker commands.

Notes : JCL

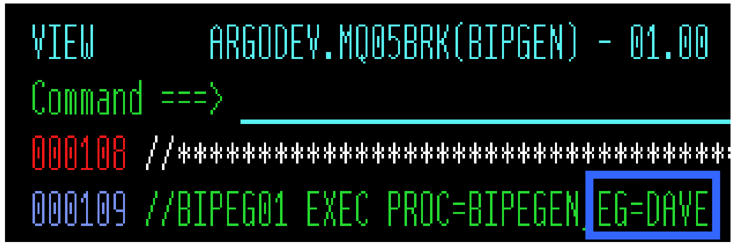
- All JCL for a specific broker is copied to the broker's component dataset.
- The JCL is customized using BIPEDIT for a specific broker.
- JCL exists to run all broker commands, such as mqsicreatebroker and mqsilist etc…

Execution Group Specific Profiles



```
VIEW          ARGODEV.MQ05BRK                              Row 00001 of 00008
Command ===>                                                   Scroll ===> CSR
            Name     Prompt        Size  Created        Changed               ID
_________   BIPBPROF                280  2011/06/22  2011/06/22 11:15:49  GORMAND
_________   BIPCRBK                 106  2011/06/22  2011/06/22 11:19:23  GORMAND
_________   BIPDSNAO
_________   BIPEDIT                  71  2011/06/22  2011/06/22 11:15:20  GORMAND
_________   BIPEGEN                   1  2011/07/15  2011/07/15 12:16:30  GORMAND
_________   BIPGEN                  111  2011/06/22  2011/06/22 11:15:36  GORMAND
_________   DAVE                      1  2011/07/15  2011/07/15 12:16:22  GORMAND
_________   MQ05BRK                 299  2011/06/22  2011/06/22 11:16:18  GORMAND
```

- There is a base profile for each broker. This is used by default for all broker address spaces.
- Execution Group specific profiles can also be specified.
- Change BIPGEN to create additional EG specific profiles

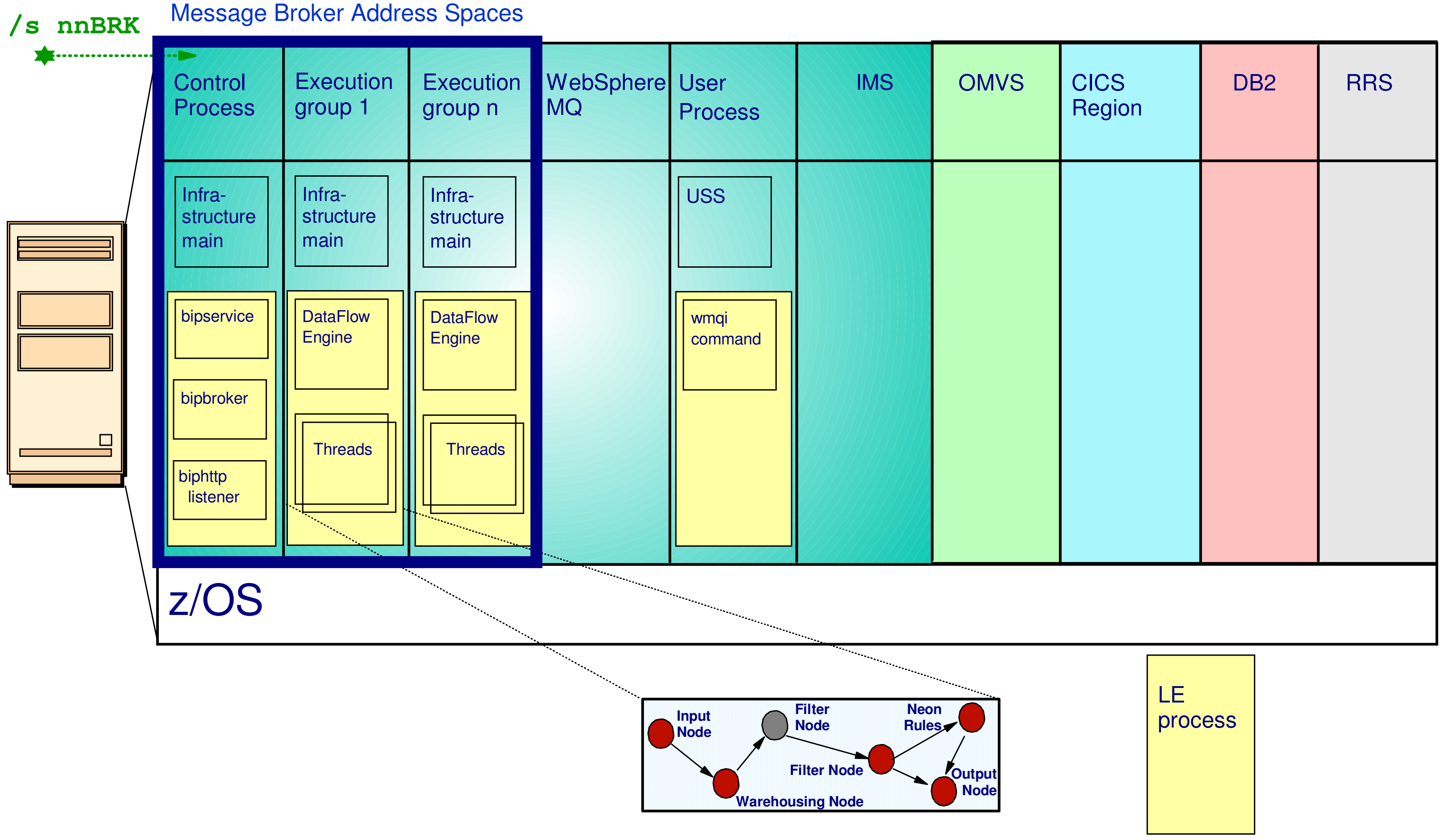


```
VIEW       ARGODEV.MQ05BRK(BIPGEN) - 01.00
Command ===>
000108 //***********************************
000109 //BIPEG01 EXEC PROC=BIPEGEN,EG=DAVE
```

Notes : Execution Group Specific Profiles

- By default Execution Groups run with the normal broker profile.
- However, each Execution Group address space can run with a customised profile.
- The Execution Group specific profile is created by customizing a new BIPEPROF member in the broker's component dataset, modifying and submitting BIPGEN.
- This produces an ENVFILE specific to the EG, and can include anything that can be specified as an environment variable.
- Displaying the broker in SDSF tells you which EG address spaces are running with specific profiles or the broker default profile.
- The Message Broker infocentre explains how to configure this.

Starting the broker



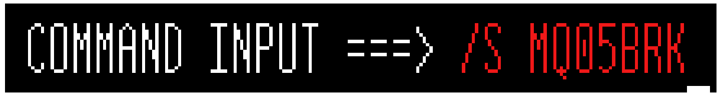
Notes : Control Address Space

- First address space started per broker.
- Contains the bipservice process. This includes the console listener for commands such as /F (modify) and /P (stop).
- Contains the bipbroker (AdminAgent) process. This starts and monitors DataFlowEngine (Execution Groups) processes. Also includes the Deployment Manager which handles requests from the Message Broker toolkit, Message Broker Explorer and CMP API.
- Contains the biphttplistener process which handles broker wide HTTP connections.
- Includes:
  - Log of messages from bipservice, bipbroker and biphttplistener
  - STDERR and STDOUT
  - Copy of ENVFILE and BIPDSNAO

Notes : DataFlowEngine Address Space

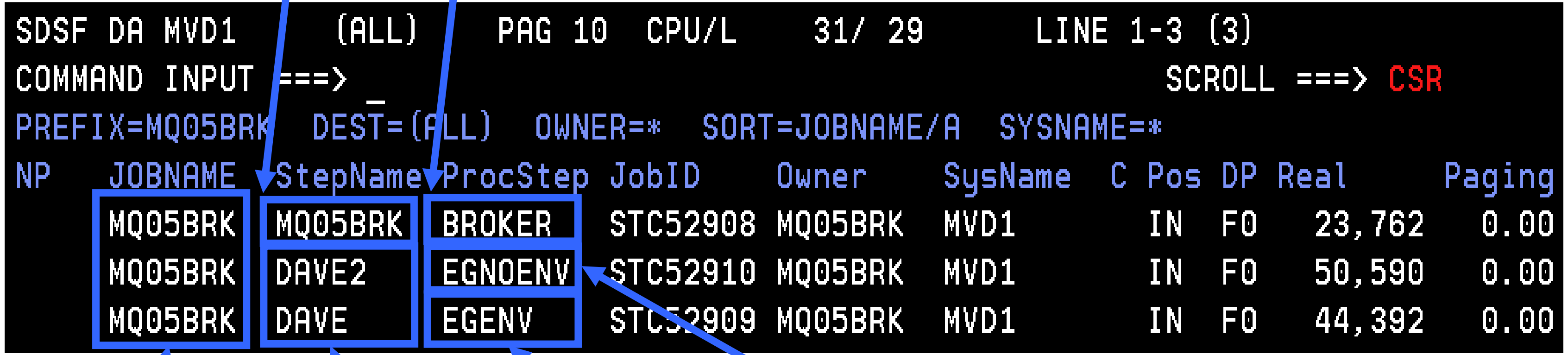
- Each Execution Group runs in its own address space.
- One or many flows can be deployed to a single Execution Group.
- Provides workload isolation and WLM configuration options.
- Each DataFlowEngine address space can run with a unique set of environment variables (for example JVM options).
- Each address space includes:
  - A log of all messages from the DataFlowEngine process
  - STDERR, STDOUT
  - Copy of the ENVFILE (environment file)
  - Copy of the DSNAOINI (ODBC configuration file)

Displaying a broker in SDSF



```
COMMAND INPUT ===> /S MQ05BRK
```

StepName same as JOBNAME and ProcStep BROKER, identifies this as the control address space.



```
SDSF DA MVD1     (ALL)    PAG 10  CPU/L    31/ 29      LINE 1-3 (3)
COMMAND INPUT ===> _                                  SCROLL ===> CSR
PREFIX=MQ05BRK  DEST=(ALL)  OWNER=*  SORT=JOBNAME/A  SYSNAME=*
NP   JOBNAME  StepName ProcStep JobID    Owner    SysName  C Pos DP Real      Paging
     MQ05BRK  MQ05BRK  BROKER   STC52908 MQ05BRK  MVD1       IN  F0   23,762   0.00
     MQ05BRK  DAVE2    EGNOENV  STC52910 MQ05BRK  MVD1       IN  F0   50,590   0.00
     MQ05BRK  DAVE     EGENV    STC52909 MQ05BRK  MVD1       IN  F0   44,392   0.00
```

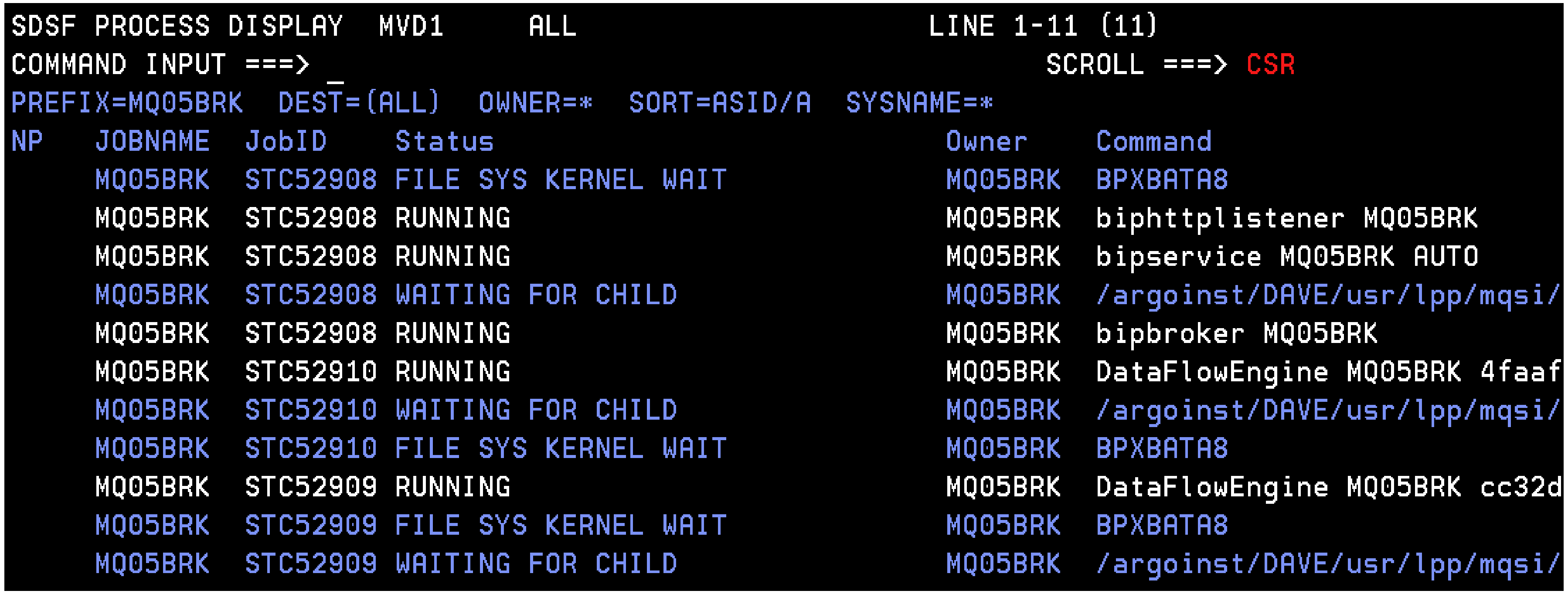
Broker JOBNAME. All address spaces associated with a single broker have the same name!

Execution Groups, identified by name! (last 8 chars of EG name)

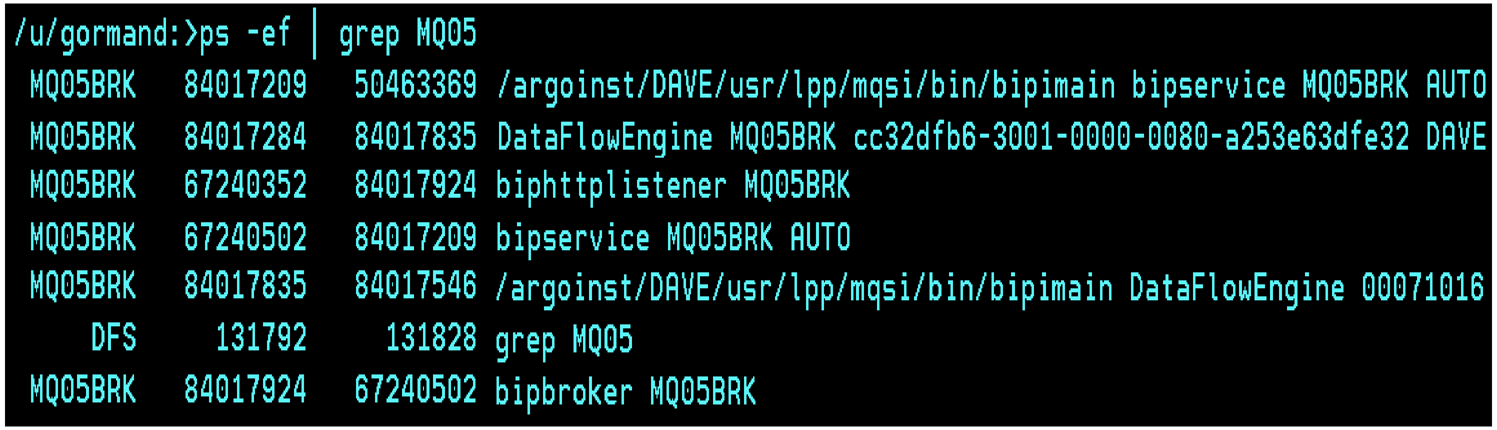
EGENV identifies this address space as running with an EG specific profile!

EGNOENV identifies this address space as running with just the broker wide profile!

Displaying broker at the process level



```
SDSF PROCESS DISPLAY  MVD1     ALL                             LINE 1-11 (11)
COMMAND INPUT ===> _                                            SCROLL ===> CSR
PREFIX=MQ05BRK  DEST=(ALL)  OWNER=*  SORT=ASID/A  SYSNAME=*
NP   JOBNAME   JobID    Status                            Owner    Command
     MQ05BRK   STC52908 FILE SYS KERNEL WAIT              MQ05BRK  BPXBATA8
     MQ05BRK   STC52908 RUNNING                           MQ05BRK  biphttplistener MQ05BRK
     MQ05BRK   STC52908 RUNNING                           MQ05BRK  bipservice MQ05BRK AUTO
     MQ05BRK   STC52908 WAITING FOR CHILD                 MQ05BRK  /argoinst/DAVE/usr/lpp/mqsi/
     MQ05BRK   STC52908 RUNNING                           MQ05BRK  bipbroker MQ05BRK
     MQ05BRK   STC52910 RUNNING                           MQ05BRK  DataFlowEngine MQ05BRK 4faaf
     MQ05BRK   STC52910 WAITING FOR CHILD                 MQ05BRK  /argoinst/DAVE/usr/lpp/mqsi/
     MQ05BRK   STC52910 FILE SYS KERNEL WAIT              MQ05BRK  BPXBATA8
     MQ05BRK   STC52909 RUNNING                           MQ05BRK  DataFlowEngine MQ05BRK cc32d
     MQ05BRK   STC52909 FILE SYS KERNEL WAIT              MQ05BRK  BPXBATA8
     MQ05BRK   STC52909 WAITING FOR CHILD                 MQ05BRK  /argoinst/DAVE/usr/lpp/mqsi/
```



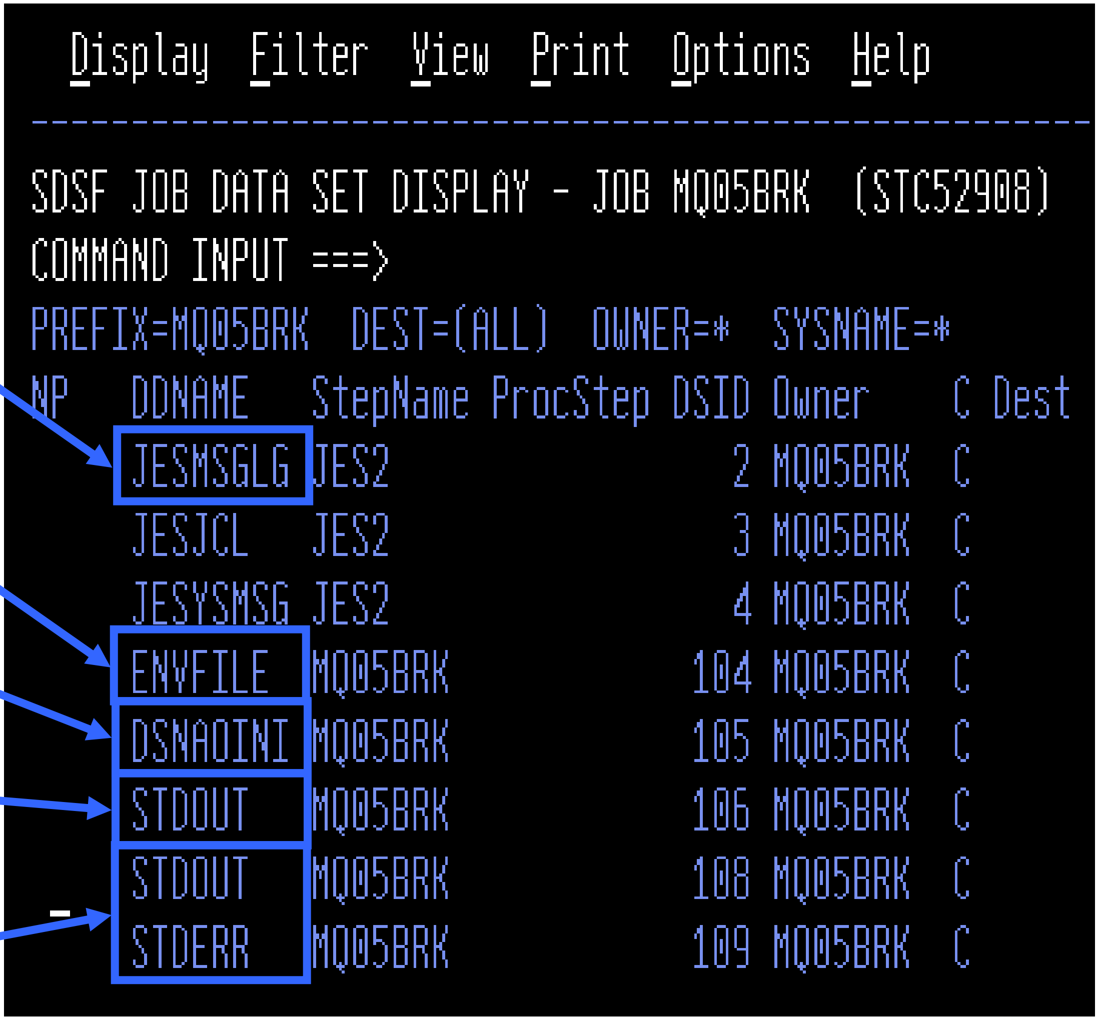
```
/u/gormand:>ps -ef | grep MQ05
 MQ05BRK  84017209  50463369 /argoinst/DAVE/usr/lpp/mqsi/bin/bipimain bipservice MQ05BRK AUTO
 MQ05BRK  84017284  84017835 DataFlowEngine MQ05BRK cc32dfb6-3001-0000-0080-a253e63dfe32 DAVE
 MQ05BRK  67240352  84017924 biphttplistener MQ05BRK
 MQ05BRK  67240502  84017209 bipservice MQ05BRK AUTO
 MQ05BRK  84017835  84017546 /argoinst/DAVE/usr/lpp/mqsi/bin/bipimain DataFlowEngine 00071016
     DFS    131792    131828 grep MQ05
 MQ05BRK  84017924  67240502 bipbroker MQ05BRK
```

Notes : Displaying broker at the process level

- Displaying a broker in SDSF.PS or in USS shows the different processes running inside the individual address spaces.
- On z/OS, a special process called bipimain is started first in all address spaces. This performs authorised functions.
- Bipservice is started in the Control address space. It contains the console listener, and starts bipbroker.
- Bipbroker (also known as the AdminAgent) starts and monitors the DataFlowEngine processes and the biphttplistener process. It is the AdminAgent which the MB toolkit, MB Explorer, commands and CMP API applications connect to.
- The biphttplistener process is the broker wide HTTP listener.
- The DataFlowEngine process is the Execution Group in which the Message Flows run.

Broker JOB output

```
 ?     MQ05BRK  DAVE     EGENV
```



```
  Display  Filter  View  Print  Options  Help
 ----------------------------------------------------------
 SDSF JOB DATA SET DISPLAY - JOB MQ05BRK  (STC52908)
 COMMAND INPUT ===>
 PREFIX=MQ05BRK  DEST=(ALL)  OWNER=*  SYSNAME=*
 NP    DDNAME   StepName ProcStep DSID Owner    C Dest
       JESMSGLG JES2                  2 MQ05BRK  C
       JESJCL   JES2                  3 MQ05BRK  C
       JESYSMSG JES2                  4 MQ05BRK  C
       ENVFILE  MQ05BRK             104 MQ05BRK  C
       DSNAOINI MQ05BRK             105 MQ05BRK  C
       STDOUT   MQ05BRK             106 MQ05BRK  C
       STDOUT   MQ05BRK             108 MQ05BRK  C
       STDERR   MQ05BRK             109 MQ05BRK  C
```

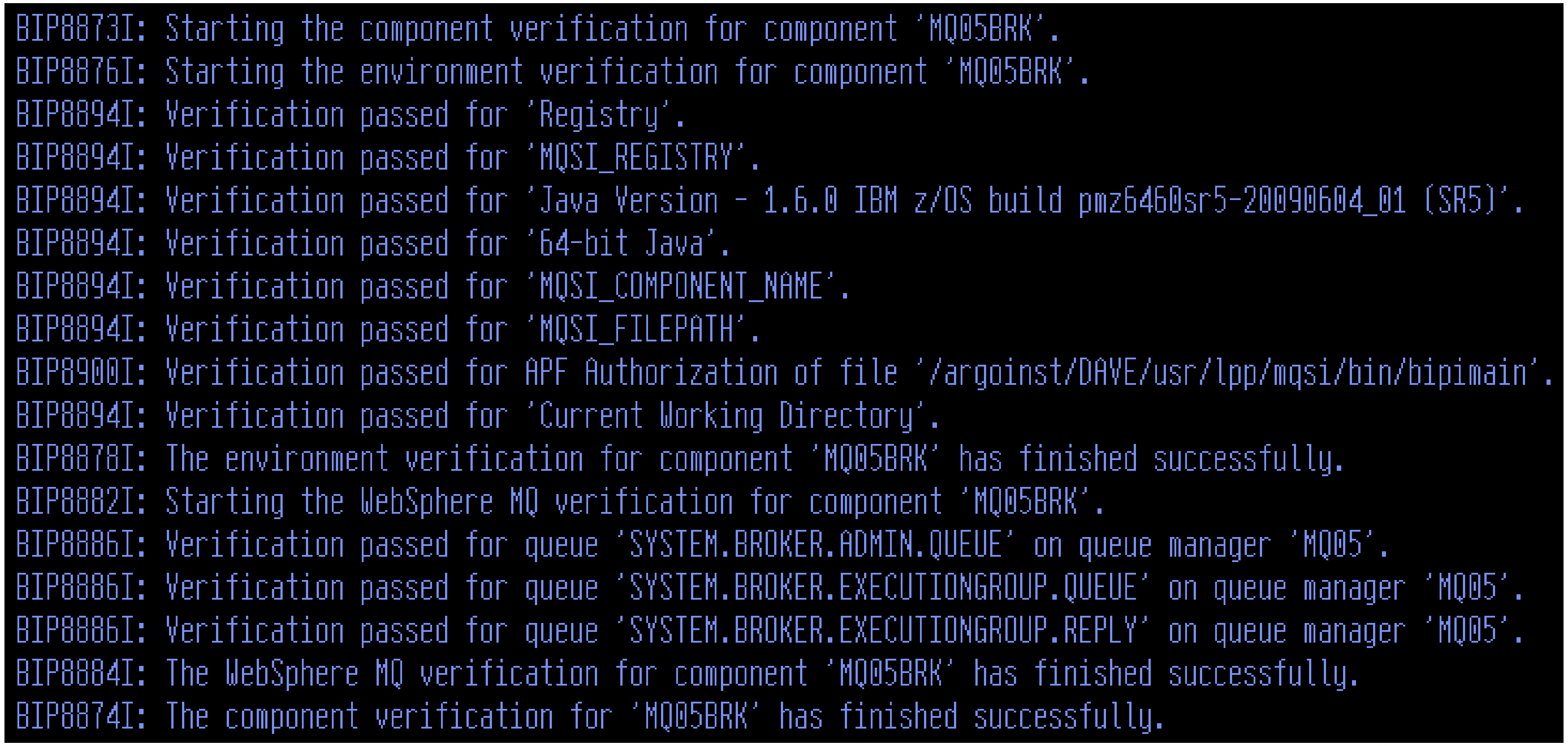
Broker BIP messages
Environment file for this address space
ODBC configuration
Output from verification STEP (control address space only)
STDERR/STDOUT from broker processes

Notes : Broker JOB output

- A broker JOB includes any BIP product messages and anything written to STDOUT and STDERR by the processes contained within the address space.
- The ENVFILE and DSNAOINI (BIPDSNAO in the component’s dataset) are also copied for a complete record.

Broker Verification

- When a broker starts, the control address runs a verification STEP checking the broker configuration and environment.

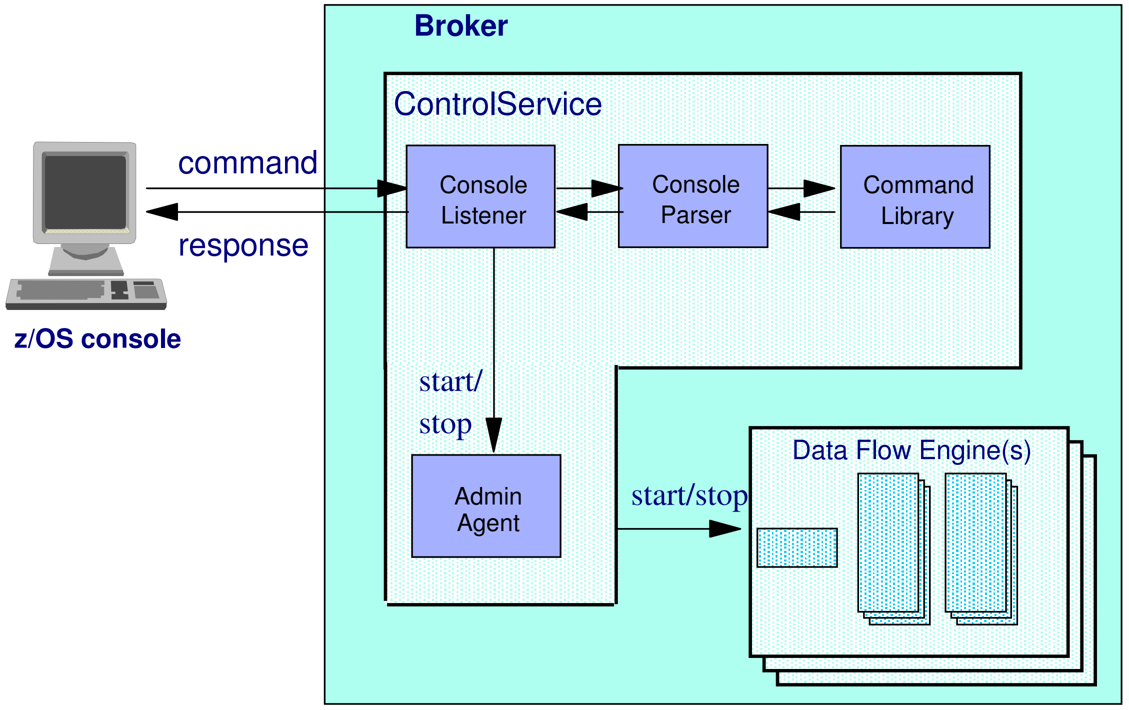


```
BIP8873I: Starting the component verification for component 'MQ05BRK'.
BIP8876I: Starting the environment verification for component 'MQ05BRK'.
BIP8894I: Verification passed for 'Registry'.
BIP8894I: Verification passed for 'MQSI_REGISTRY'.
BIP8894I: Verification passed for 'Java Version - 1.6.0 IBM z/OS build pmz6460sr5-20090604_01 (SR5)'.
BIP8894I: Verification passed for '64-bit Java'.
BIP8894I: Verification passed for 'MQSI_COMPONENT_NAME'.
BIP8894I: Verification passed for 'MQSI_FILEPATH'.
BIP8900I: Verification passed for APF Authorization of file '/argoinst/DAVE/usr/lpp/mqsi/bin/bipimain'.
BIP8894I: Verification passed for 'Current Working Directory'.
BIP8878I: The environment verification for component 'MQ05BRK' has finished successfully.
BIP8882I: Starting the WebSphere MQ verification for component 'MQ05BRK'.
BIP8886I: Verification passed for queue 'SYSTEM.BROKER.ADMIN.QUEUE' on queue manager 'MQ05'.
BIP8886I: Verification passed for queue 'SYSTEM.BROKER.EXECUTIONGROUP.QUEUE' on queue manager 'MQ05'.
BIP8886I: Verification passed for queue 'SYSTEM.BROKER.EXECUTIONGROUP.REPLY' on queue manager 'MQ05'.
BIP8884I: The WebSphere MQ verification for component 'MQ05BRK' has finished successfully.
BIP8874I: The component verification for 'MQ05BRK' has finished successfully.
```

Notes : Broker Verification

- When a broker is first started, a verification program runs in the control address space.
- This checks that broker has the correct level of Java defined, that it can access the necessary files, and that bipimain is APF authorised, among many other checks.
- If a problem is found, the error is reported in the JOBLOG and the broker is not started.

Console commands



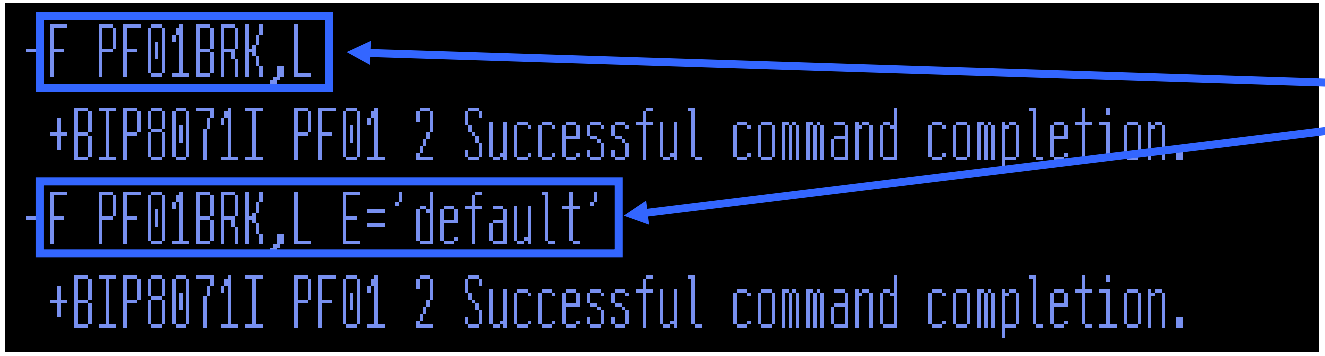
Short	Long
SC	Start component
PC	Stop component
CT	Change trace
RT	Report trace
L	List
RE	Reload
CB	Change broker
CS	Change flow stats
RS	Report flow stats
DP	Deploy
CX	Change flow user exits
RX	Report flow user exits
RC	Reload security
CM	Change flow monitoring
RM	Report flow monitoring
CR	Change resource stats
RR	Report resource stats

Start a broker	/S <Broker>
Stop a broker	/P <Broker>
Modify a broker	/F <Broker>,cmd

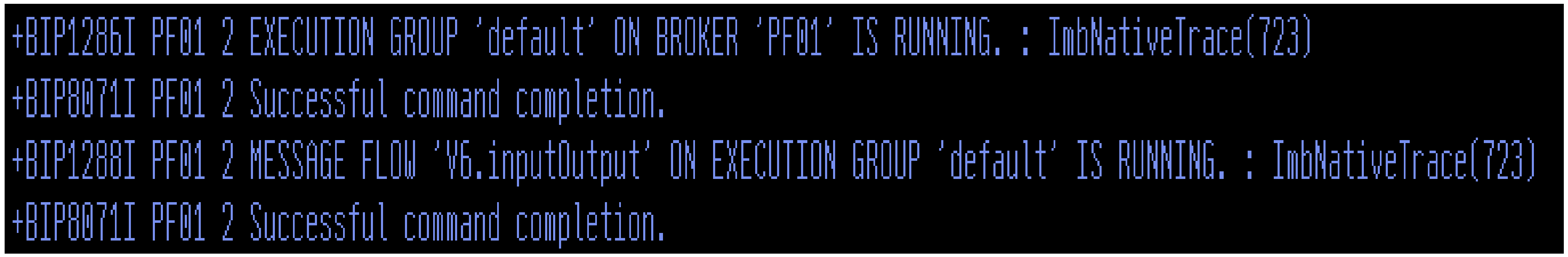
Notes : Console commands

- A number of Message Broker commands can be entered directly from the console.
- The commands are listed on the previous slide.
- Administrative changes can also be made using JCL, the MB Explorer or the CMP API.

Console command output



```
-F PF01BRK,L
 +BIP8071I PF01 2 Successful command completion.
-F PF01BRK,L E='default'
 +BIP8071I PF01 2 Successful command completion.
```

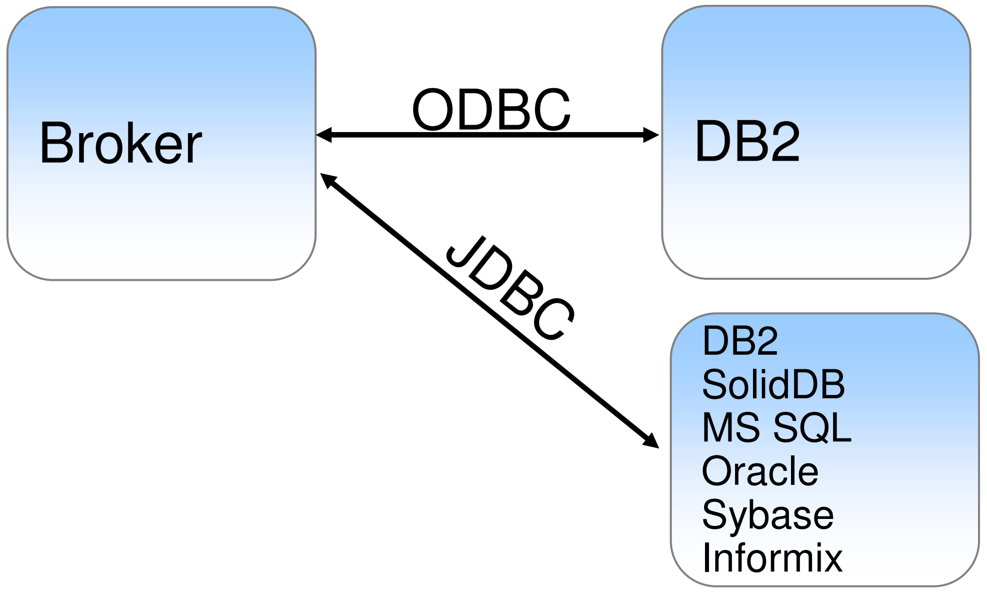


```
+BIP1286I PF01 2 EXECUTION GROUP 'default' ON BROKER 'PF01' IS RUNNING. : ImbNativeTrace(723)
+BIP8071I PF01 2 Successful command completion.
+BIP1288I PF01 2 MESSAGE FLOW 'V6.inputOutput' ON EXECUTION GROUP 'default' IS RUNNING. : ImbNativeTrace(723)
+BIP8071I PF01 2 Successful command completion.
```

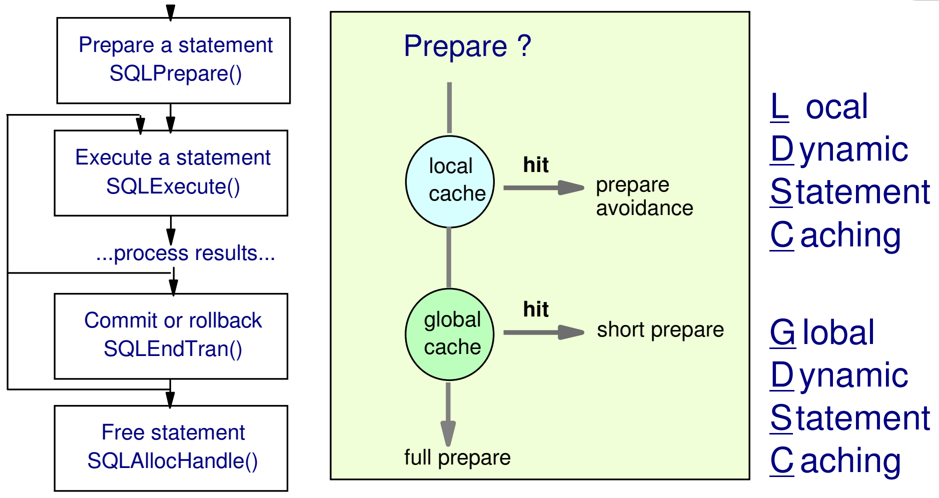
Notes : Console command output

- Console command output is written to the control address space .

Database Connectivity on z/OS



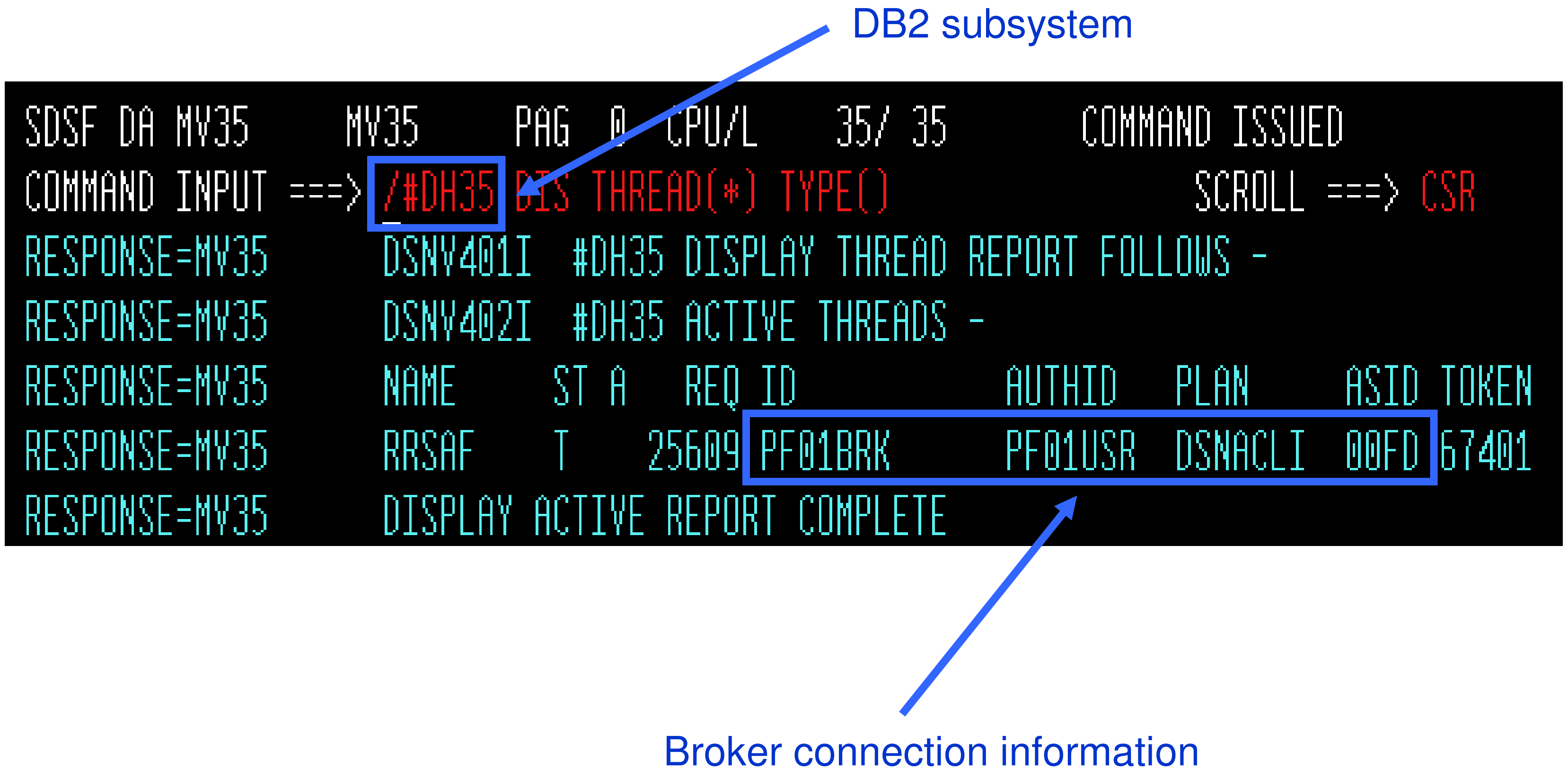
- Easy access to user databases
- Access DB2 via ODBC
  - *full statement caching and prepare avoidance is used to give high performance access.*
- Access DB2 and other Databases via JDBC



Notes : Database Connectivity

- **What is Open Database Connectivity (ODBC)?**
  - You may well be used to using Structured Query Language (SQL) to write programs that access and update data in a database.
  - SQL does not have a call level interface, rather it is a declarative language which is translated by means of a product specific preprocessor which changes the SQL statements into product specific calls to a database plan. This plan is a representation of the access your program requires to the database.
  - You're probably also aware that even though SQL is standard, the process of translating and binding SQL is not! ODBC is designed to address this problem. It is important to WebSphere brokers, because uniform data source access on any platform accessible to supports Compute, Filter and Database nodes for routing, warehousing and data enrichment without product specific SQL preprocessing.
- **Benefits of using ODBC.**
  - ODBC SQL is not precompiled or bound; it uses standard functions to execute SQL statements and related services at run time.
  - ODBC enables you to write portable applications that are independent of any particular database product translation or binding scheme. This independence means applications do not have to be recompiled or rebound to access different data sources, but rather just connect to the appropriate data source at run time.
  - One of the disadvantages of ODBC is that plans have to be generated at runtime, and are invalidated after commit processing. As the majority of cost is associated with the plan, ODBC is as expensive as dynamic SQL. We'll see how this cost can be alleviated.
- **ODBC transaction processing model.**
  - A program using OBDC, and your deployed dataflow is just that (!), takes the SQL statement (issued by the filter, compute, database nodes) and prepares it using the SQLPrepare call. This is exactly analogous to the binding process that static SQL has - it's just that the plan is built at run time.   (Configure ODBC tracing in the BIPDSNAO file to observe application behaviour!)
  - The prepared statement is then executed as part of a transactional unit of work, and the results processed. The statement can be re-executed before the transaction is committed and this greatly improves performance. However, once the transaction has been committed, along as you're using local and dynamic statement caching, this prepare is only performed once.
  - If you're using DB2, use GDSC (ZPARM CACHEDYN=YES) to improve the performance of ODBC type applications by caching prepared statements in using a skeleton statement kept beyond transaction commit. The broker uses Local Statement Caching for prepare avoidance, significantly improving database performance, bringing it close to static SQL.
- **Check SOE for DB2 APARs required for 64bit ODBC driver!**

Displaying broker threads in DB2



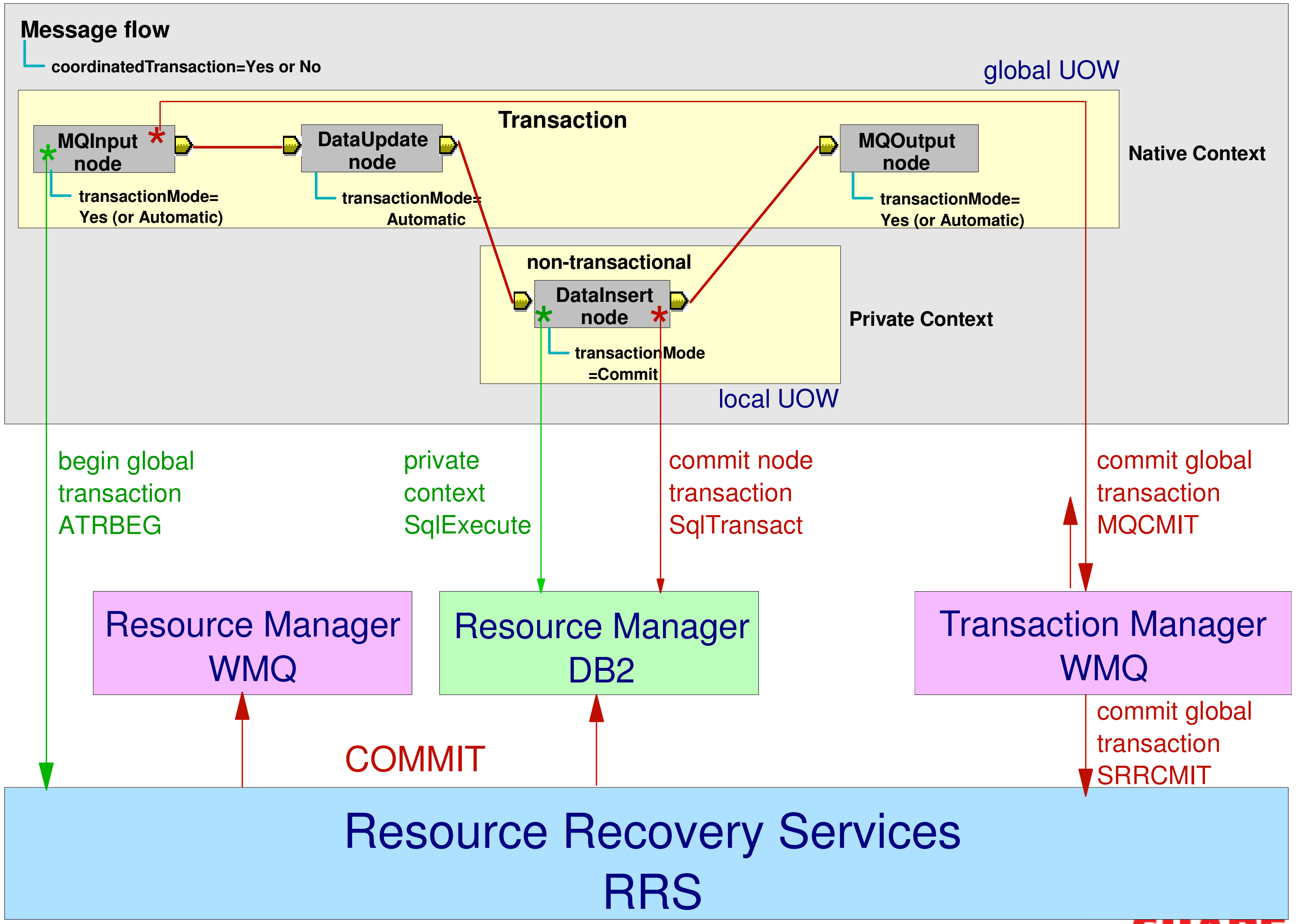
```
SDSF DA MV35     MV35     PAG  0  CPU/L    35/ 35      COMMAND ISSUED
COMMAND INPUT ===> /#DH35 DIS THREAD(*) TYPE()                SCROLL ===> CSR
RESPONSE=MV35      DSNV401I  #DH35 DISPLAY THREAD REPORT FOLLOWS -
RESPONSE=MV35      DSNV402I  #DH35 ACTIVE THREADS -
RESPONSE=MV35      NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
RESPONSE=MV35      RRSAF    T     25609 PF01BRK     PF01USR  DSNACLI  00FD 67401
RESPONSE=MV35      DISPLAY ACTIVE REPORT COMPLETE
```

Notes : Displaying broker threads in DB2

- By displaying active threads in the DB2 subsystem, we can see that the broker has a connection.
- This shows RRS as the transaction coordinator, it also shows the USERID and PLAN.

Transaction Model

The z/OS broker has a global transaction model exactly as you'd expect. It is possible for nodes to elect to commit outside this transaction. RRS is used for context management & commitment control between the flows resource managers, but only when required.


Message flow
coordinatedTransaction=Yes or No
global UOW
Transaction
MQInput node
transactionMode= Yes (or Automatic)
DataUpdate node
transactionMode= Automatic
MQOutput node
transactionMode= Yes (or Automatic)
Native Context
non-transactional
DataInsert node
transactionMode =Commit
Private Context
local UOW
begin global transaction ATRBEG
private context SqlExecute
commit node transaction SqlTransact
commit global transaction MQCMIT
Resource Manager WMQ
Resource Manager DB2
Transaction Manager WMQ
commit global transaction SRRCMIT
COMMIT
Resource Recovery Services RRS

Notes : Transaction Model

- **Transactional message flows are important**
  - A message flow which transforms and routes data often has a need to be transactional. That is, the message flow must complete either *all or none* of its processing. Remember, from an end-to-end application perspective, the message flow is *part* of the application.
- **Transactional data flows and data nodes.**
  - A message flow can be identified as transactional using the Coordinated Transaction checkbox on a broker assigned message flow. The intention behind this attribute is that all node operations within the message flow can be coordinated under the same, global, transaction. On z/OS, this option is always used for message flows, whether selected or not.
  - A node performs its operations within the envelope of this message flow global transaction, and can elect to be within the global transaction or not. A Transaction Mode checkbox enables this for WMQ and database nodes. Note the visibility (ACID) implications!
- **Resource Recovery Services (RRS) is *NOT* always the transaction coordinator.**
  - As message flows run in essentially a batch type address spaces, RRS is the global transaction coordinator, if required.
  - Execution groups are linked with an MQ RRS stub, so WMQ registers an interest with RRS for commitment control.
  - Specifying the keywords CONNECTTYPE=2, AUTOCOMMIT=0, MULTICONTEXT=0, and MVSATTACHTYPE=RRSAF in the initialization file BIPDSNAO enables global transaction processing.
- **RRS Context**
  - RRS Context is a concept that enables a program to have different roles. It's like one person having many ways of behaving which don't interact with each other. It means that applications can simultaneously be different things to different systems.
  - Broker flows have two contexts. A *native* context is used whenever it wants to perform the role of including node operations under the global transaction. A *private* one has the effect of excluding database node operations from a global transaction.
  - Plug-in nodes are always within the global transaction. A message flow is always in *native* context for these nodes.
- **WebSphere MQ**
  - *Transaction Mode* within a message queuing node governs whether MQPUT and MQGET operations are explicitly performed either inside or outside syncpoint on a per call basis. These nodes therefore always use the native, and never the private, RRS context.
- **Database**
  - For database nodes, *Transaction Mode* determines under which RRS context the transaction will be performed. If the node is within the global transaction, then the native context is used. For a Transaction Mode of *commit*, the private context, so that DB2 and RRS see the operation from a logically different party. These nodes commit (using SQLTransact) their operations as the node is exited.
- **Commitment Control**
  - The global transaction is begun implicitly when a resource manager communicates with RRS. The overall message transaction is committed (or backed out!) control returns to the input node. At COMMIT time, WMQ will pass control to RRS only if required.
  - RRS will call all registered resource managers (WMQ, DB2) in a two phase commit protocol to ensure a global transaction. Recall that nodes which elected for a Transaction Mode of commit had resources updated (and externally visible!) close to their point of issuing. If RRS is not required WMQ will perform the commitment control and delete any RRS interests.

Agenda

- Message Broker Overview
- Message Broker on z/OS.
- Value of Message Broker on z/OS.
  - High Availability
  - WLM
  - Accounting and Chargeback
  - Reduced TCO
  - z/OS specific nodes

High Availability

- **Parallel SYSPLEX**
  - Message flows deployed to brokers on z/OS can benefit from using resource managers such as WebSphere MQ (Shared Queues) and DB2 (Data Sharing Group) which are Parallel SYSPLEX aware.
- **Automatic Restart Manager (ARM)**
  - ARM is a z/OS subsystem which ensures appropriately registered applications and subsystems can be restarted after a failure.
  - Applications and subsystems can either be restarted on the image on which they failed, or in the case of image failure, on another available image in the SYSPLEX.
- **WebSphere MQ Clustering**
  - For z/OS users not using Parallel SYSPLEX, MQ clustering provides an excellent way to exploit high availability for new messages in the event of failure.
  - Although MQ clustering is a “shared nothing solution”, in many scenarios it can be good enough to enable the availability of new work, especially when combined with ARM.

Notes : High Availabilty

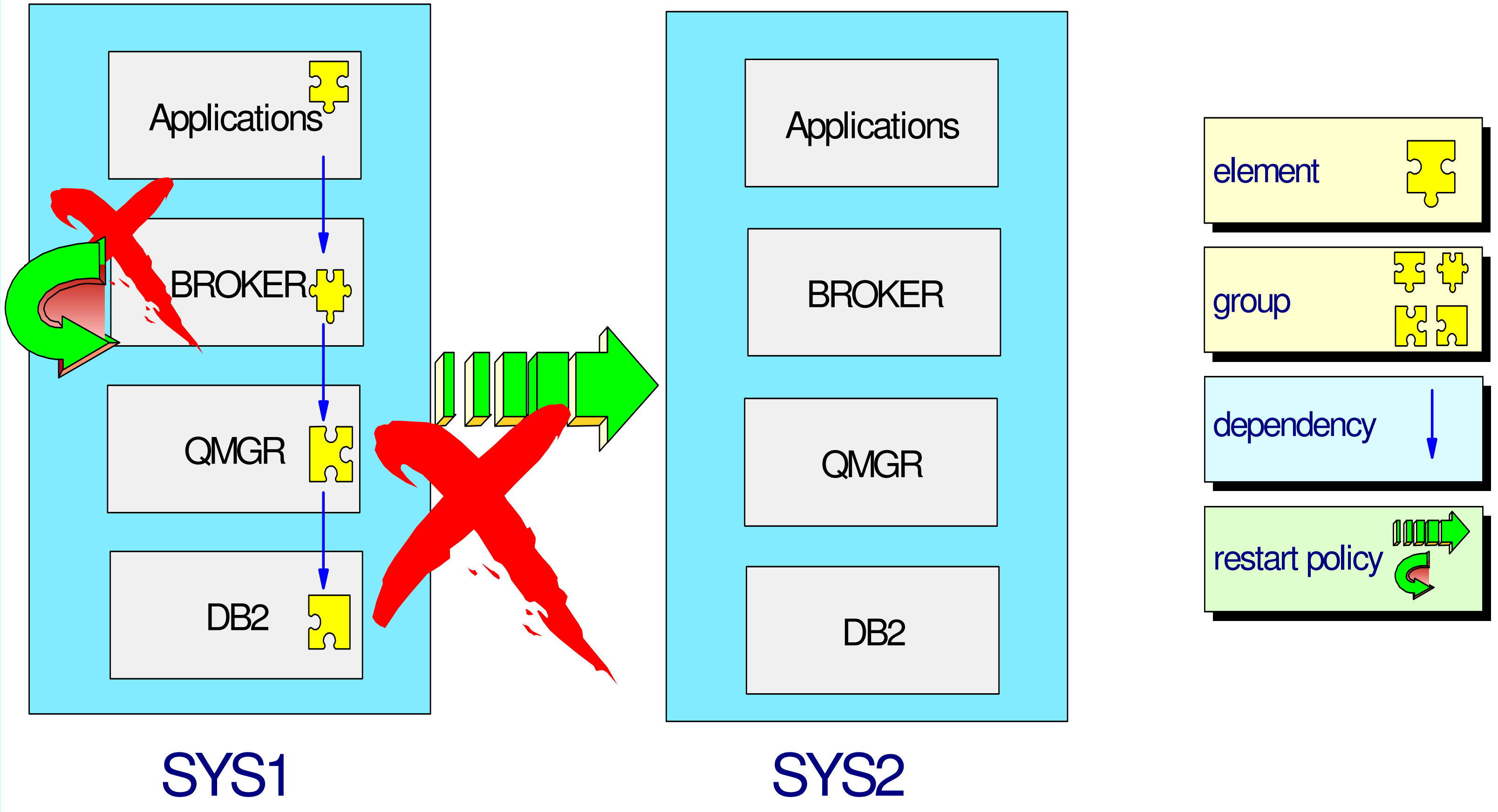
- Parallel SYSPLEX.
  - Message flows deployed to brokers on z/OS can benefit from using resource managers such as WebSphere MQ and DB2 which are Parallel SYSPLEX aware.
  - Objects including WebSphere MQ queues and DB2 databases are available as a shared resource within a SYSPLEX, making them simultaneously available to all applications within the SYSPLEX. As long as one z/OS image is available these resources are available for use.
  - What's really powerful though, is the fact that this is a truly shared resource, and in the event of failure, the remaining images will make the failed work available immediately, without the need for restart. This is high availability in its highest sense.
  - For example, a message flow deployed to brokers in a WebSphere MQ *Queue Sharing Group* transforming XML messages from a shared input queue, may perform that processing *anywhere* in the SYSPLEX.
  - As long as one queue manager in the group remains available, messages can continue to be transformed.
  - What differentiates this from an availability perspective over “shared nothing” type solutions is that after a failure, rolled back messages are made available immediately within the SYSPLEX such that they can be processed by other flow instances servicing the shared input queue.
  - Similar capabilities exist for DB2 resources when using DB2 *Data Sharing Groups*.

- Automatic Restart Manager (ARM).
  - ARM is a z/OS subsystem which ensures appropriately registered applications and subsystems can be restarted after a failure.
  - Applications and subsystems can either be restarted on the image on which they failed, or in the case of image failure, on another available image in the SYSPLEX.
  - Message Broker is fully ARM enabled.
  - It is simple for an operator to assign an ARM classification and name to the broker. When the broker started or stopped, it will perform the necessary registration with ARM, enabling it to be restarted as defined in the ARM policy by the z/OS systems programmer.

- WebSphere MQ Clustering
  - For z/OS users not using Parallel SYSPLEX, MQ clustering provides an excellent way to exploit high availability for new messages in the event of failure.
  - In this mode of operation, brokers in a domain have a partitioned copy of the input queue, and clustering ensures that in the event of a failure, new messages are sent only to queues which are still available.
  - Although MQ clustering is a “shared nothing solution”, in many scenarios it can be good enough to enable the availability of new work, especially when combined with ARM.

Restart Management using ARM

Restart is defined by the Systems Administrator to organize applications into dependent groups for recovery after failure. Failures can include application failure or system failure and restart can be performed in place or on an adjacent Sysplex image.

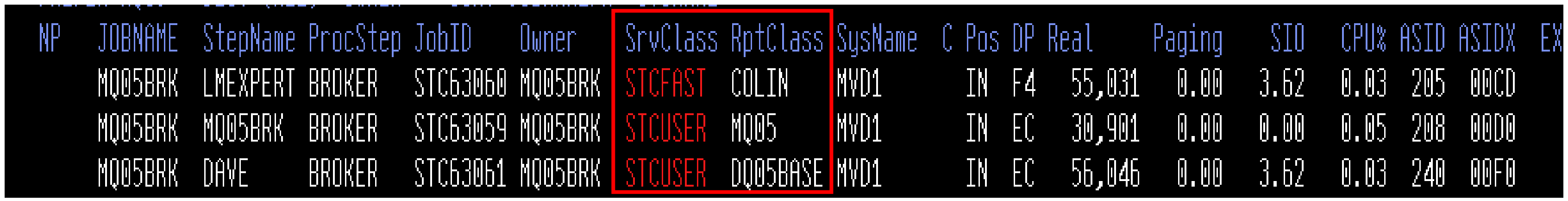


Notes : Restart Management using ARM

- **What is Automatic Restart Manager (ARM)?**
  - ARM is part of z/OS XCF and is used to restart failed address spaces after their failure or the failure of an image upon which they were executing. Subsequently, they can be either restarted on the same image, or another, if this has failed.
  - ARM therefore improves the availability of specific batch jobs or started tasks.
  - The design of MB restart is to make only the Control Process address space ARM restartable. The hierarchical relationship between the Control Process address space and in turn, its relationship to the Execution Groups means that this is sufficient.
- **Elements and Restart Groups.**
  - An address space that is using ARM services is referred to as an "element." The systems programmer, rather than the broker, defines a restart group as a set of named elements that have affinities and must be restarted together in the event of a system failure. The restart group also governs the restart sequence of elements, so that subsystem dependencies can be modelled and enforced in terms of subsystem restart.
  - An element is comprised of an 8 character type and a 16 character name. For MB, ARM usage is indicated and configured in the broker's profile BIPBPROF. For example USE_ARM=YES, ARM_ELEMENTNAME='MQ03BRK' and ARM_ELEMENTTYPE='SYSWMQI'.
- **How to Use and Configure.**
  - The ARM couple data set is the repository of the ARM policies of an installation and also of the specifics of elements with ARM status. This data set is separate from other couple data sets (e.g., those of workload manager), may have an alternate data set, and must be connected to all systems where registration and restart might occur. The ARM policy is a set of instructions from an installation about how and where (and whether) restarts are to be done. The main purpose of a policy is to define the elements comprised by a group, with particulars about dependencies in the group, overriding sources of restart techniques and parameters, selection criteria in cross-system restarts, and pacing of restarts.
  - When you configure a broker to use ARM, you'll probably want to make it dependent on many prerequisite address spaces. These include WMQ and DB2, as their operation is vital to the broker.
  - In a similar way, you might like to have ARM policies for important, restartable applications that use the broker.
- **Reference.**
  - For more information on ARM, consult Sysplex Services Guide GC28-1771.

Workload Management

- **Workload Scaling**
  - z/OS SYSPLEX brings a huge amount of potential processing capacity to the message broker with up to 96 z196 processors.
  - Message flows have a built-in multiprocessing capability. Each *instance* of a message flow can execute on a separate thread (z/OS TCB) allowing the broker to easily exploit multiple CPUs
- **Workload Isolation**
  - Simply assign separate Execution Group address spaces to different types of work.
- **z/OS Individual Execution Group Workload Management**
  - Execution groups automatically assigned to **`JOBACCT`** token
  - Use WLM Classification panels to map **`JOBACCT`** to Service and Report classes



```
NP   JOBNAME  StepName ProcStep JobID    Owner    SrvClass RptClass SysName  C Pos DP Real    Paging    SIO   CPU% ASID ASIDX  EX
     MQ05BRK  LMEXPERT BROKER   STC63060 MQ05BRK  STCFAST  COLIN    MVD1       IN  F4 55,031    0.00   3.62   0.03  205  00CD
     MQ05BRK  MQ05BRK  BROKER   STC63059 MQ05BRK  STCUSER  MQ05     MVD1       IN  EC 30,901    0.00   0.00   0.05  208  00D0
     MQ05BRK  DAVE     BROKER   STC63061 MQ05BRK  STCUSER  DQ05BASE MVD1       IN  EC 56,046    0.00   3.62   0.03  240  00F0
```

Notes : Workload Management

- Goal Oriented Resource Allocation (WLM).
  - When a message broker execution group address space starts, it is assigned a JOBACCT token. This can then be classified in Workload Manager (WLM) to service and report classes. This enables systems programmers to assign different goals (typically response time) to this service class through WLM configuration choices
  - The ability to assign WLM service classes to message processing workload has two significant benefits
    - As work passes through subsystems which are WLM enabled, the service class can be maintained. Resources such as CPU and IO "follow" users' work requests to make sure these requests meet, if possible, the goals set by WLM
    - WLM classification means that in the event of resource constraint (at peak time for example), high priority workloads can receive appropriate resource to ensure that critical workloads are completed at the expense of less important work

- Workload Scaling.
  - z/OS SYSPLEX brings a huge amount of potential processing capacity to the message broker.
  - Not only are z196 processors extremely powerful on their own, but 96 can be configured in different LPARs across the SYSPLEX configuration.
  - The message broker is designed to exploit all of these capabilities without users having to explicitly design their message flows differently .
  - Message flows have a built-in multiprocessing capability.
  - Each *instance* of a message flow can execute on a separate thread (z/OS TCB) allowing the broker to easily exploit multiple CPUs.

- Workload Scaling - continued.
  - If a flow is not able to process enough messages due to CPU constraint, it is a simple step to assign more instances to the message flow to dynamically increase the number of eligible CPUs for processing. This does not involve flow design or outage .
  - And as we've seen, it's simple to scale message flows across the SYSPLEX by deploying to multiple brokers within the SYSPLEX broker domain. Again, this reconfiguration process is dynamic and does not require restart .

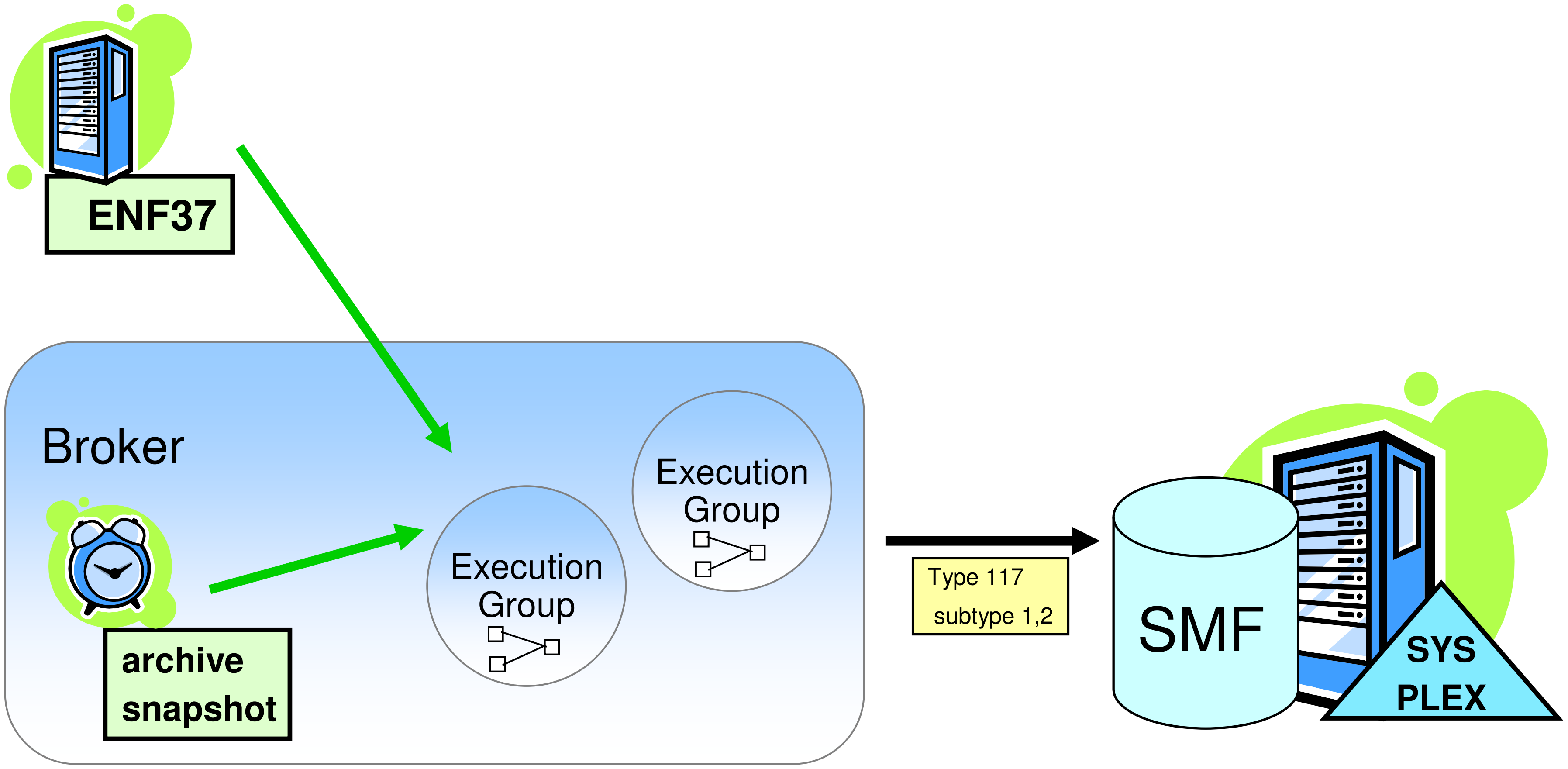
- Workload Isolation
  - From a storage isolation perspective, broker execution groups are completely separated, storage in one address space is not visible to another execution group. Moreover, if the event of a failure, an execution group is restarted without affecting any execution groups owned by the broker; failures are isolated to execution groups.

Reporting & Chargeback

- **System Management Facilities (SMF)**
  - Message Broker allows operational staff to control the gathering of SMF data on a per message flow basis. The message flow developer does not need to instrument their design in any way.
  - Enables efficient and effective tuning of message flows.
  - Allows infrastructure departments to charge users according to their utilization of broker facilities.

- **Coordinated Reporting**
  - SMF also allows subsystems to report performance and accounting information at the same time for a given processing interval.
  - Makes it possible to correlate statistics between Message Broker, MQ and DB2 subsystems and artefacts.

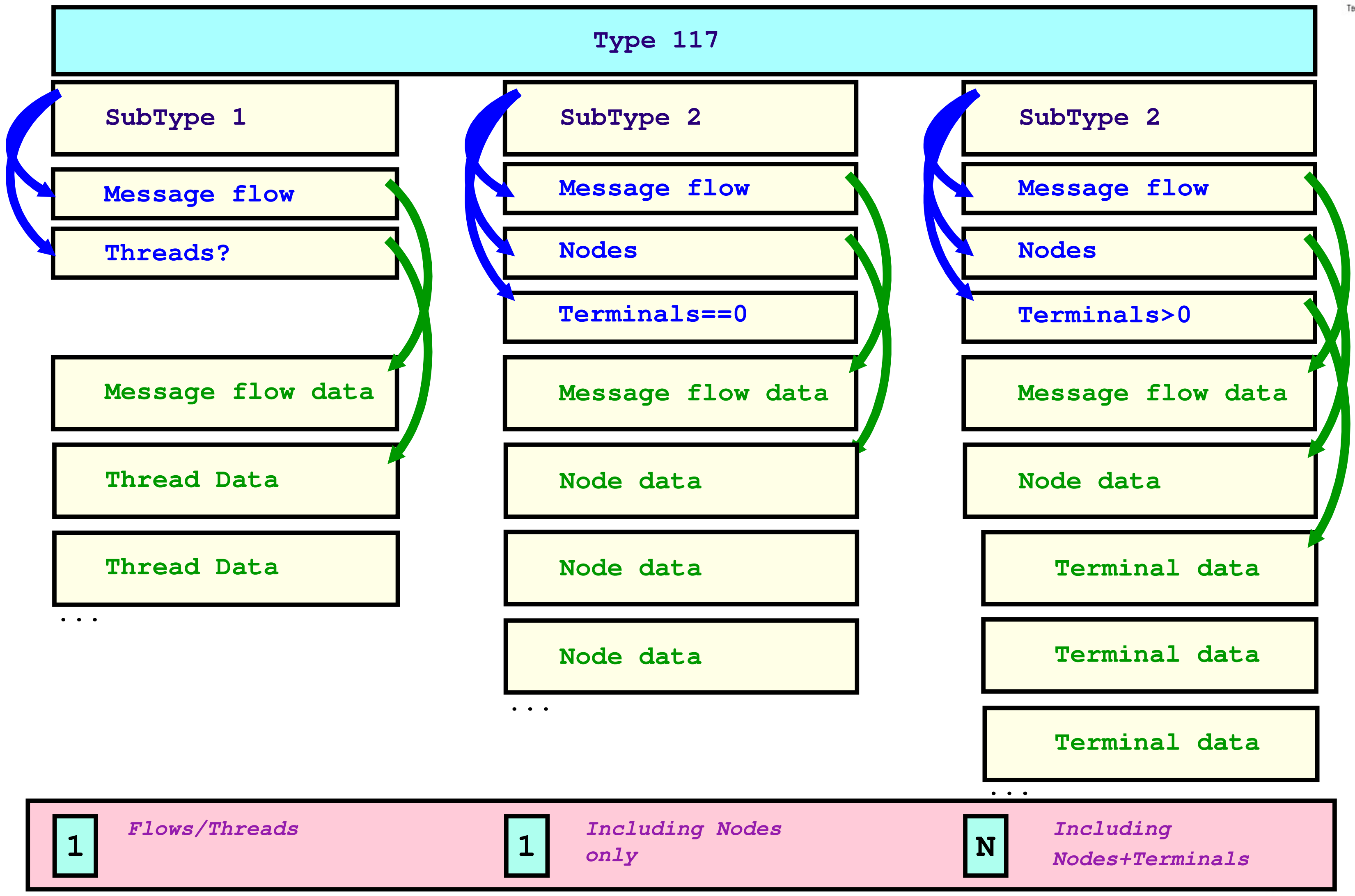
SMF output


ENF37
Broker
archive snapshot
Execution Group
Execution Group
Type 117
subtype 1,2
SMF
SYS PLEX

Notes : SMF Output

- Destinations and Collection Scope
  - This foil identies the main areas for consideration when examining accounting and statistics data
    - When is the data produced?
    - In what format is the data (UserTrace / XML / SMF)?
    - How is the data accessed?
    - What is the granularity and scope of the information that can be seen?
- Accounting and Statistics Timers
  - Information is gathered at regular intervals according to timers. There are two classes of timers, internal and external.
  - Archive and Snapshot timers are internal timers set by broker parameters which govern when these data are written to their destinations.
  - An external timer is available on z/OS, namely ENF37. This can be used to to drive SMF, UserTrace and PubSub intervals. ENF is also important to allow consolidated reporting of SMF information across major subsystems, e.g. you might coordinate queue manager and broker activity to best understand how to tune your queue manager for particular flows.
- A Variety of Output Destinations and Formats
  - It's possible to gather this information in different formats according to technology used to access it.
  - z/OS SMF: For z/OS, this option generates SMF type 117 records having subtypes 1 and 2 depending on the granularity of information requested by the user for a particular flow.
  - Publish Subscribe.
  - UserTrace.
  - You may request different output destinations for Snapshot and Archive Stats by Message Flow.
- Options for Reporting Scope and Granularity
  - z/OS SMF can be collected for any broker within a SYSPLEX. It can be integrated with all other SMF enabled z/OS products and subsystems - literally hundreds!
  - UserTrace is collected for the broker. If several brokers are operating on a single machine, then several datasets may be accessed with ease. This data is human readable and relatively useful if you're prototyping.
  - PubSub can be collected for any broker throughout the domain. This is incredibly powerful; it means you can sit anywhere in a domain and request information about particular nodes in a flow on a particular execution group on a broker on a differnt machine! OR you could put in wild card subscriptions to gather information from several different sources! XML is very structured, which makes it ideal for multi platform reporting.
  - Moreover, the fact that each report has the details of the broker and execution group in it measn that you can aggregate information as well.

z/OS SMF Record Structure



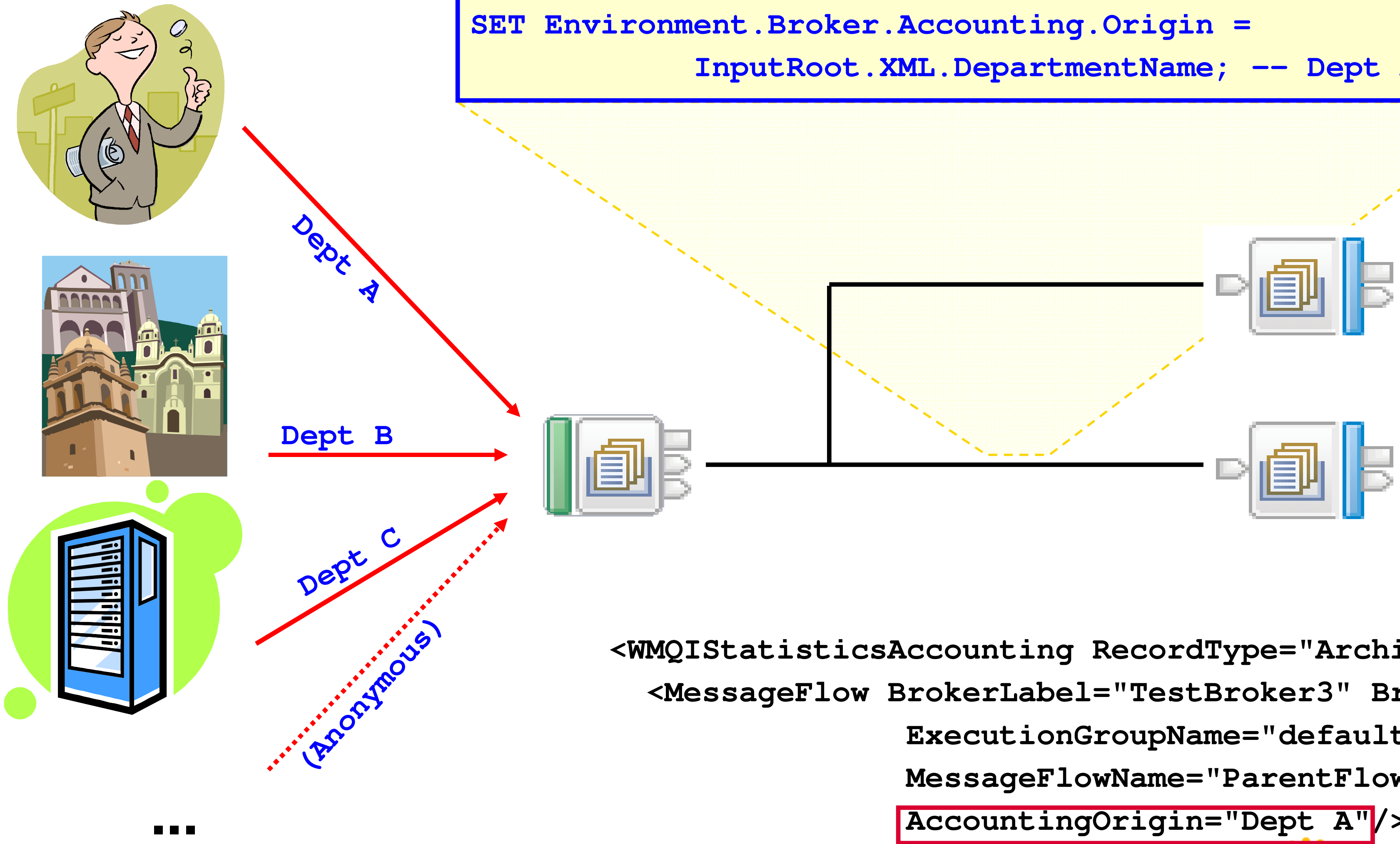
BipSmf.h

Notes : z/OS SMF Record Structure

- SMF 117 records describe the A&S information
  - The subtype of SMF 117 records written depends on the data currently being collected.
  - A type1 record is produced when a flow is only collecting message flow data or Threads data. A single type1 record is produced with all threads included.
  - Type2 records are produced when a flow is collecting nodes data. When *only* nodes data is collected, a single type 2 record is written to SMF, whereas when nodes *and* terminals are being collected, multiple type 2 records are written to SMF.
- A BipSMF.h describes the SMF records
  - You can use the sample formatter and the BipSMF.h header file to produce your own reports.
- The broker userid must be permitted to the BPX.SMF facility class profile to write SMF records.

Accounting Origin

```
SET Environment.Broker.Accounting.Origin =
            InputRoot.XML.DepartmentName; -- Dept A,B,C
```



```
<WMQIStatisticsAccounting RecordType="Archive"
  <MessageFlow BrokerLabel="TestBroker3" BrokerUUI
                     ExecutionGroupName="default" Execu
                     MessageFlowName="ParentFlow" Start
                     AccountingOrigin="Dept A"/>
    ...
```

Notes : Accounting Origin

- AccoutingOrigin Allows you to Classify Reports
  - In a consolidated flow, i.e. one being used by several different users or classes of users, it's important to be able to identify the costs associated with a particular user or class of user. From Version 5, you can request Accounting and Statistics reports to be generated to identify the originator of the input messages.
  - This allows brokers to have a minimum number of flows for accounting purposes and still chargeback, benefitting from economy of scale, and administrative burden reduction.
  - The foil shows messages originating from three different departments - A, B and C. These messages are all processed by the same message flow, but the gathered reports identify the cost breakdown by originating department.
  - If you examine the report snippet, you can see that the AccountingOrigin tag identifies the report as belonging to "Dept. A".
  - The Accounting origin is set inside the flow when it has determined the origin of the message. Any technique can be used to do determine this; for example a flow might use a field in the MQMD, or a field in the body of the message. The AccountingOrigin is completely virtual, for example it might periods of the day to allow you to profile usage with time! (You could also use archive statistics for this.)
- Messages are Classified by the Flow
  - As a message passes through a message flow, if the flow at some points decides that the message needs to be classified, it may do so.
  - It sets the **`Environment.Broker.Accounting.Origin`** tree element to identify the origin. When the message has been finished with, the data related to its processing is collected separately to messages with different AccountingOrigin.
  - Classification is usually done using User defined ESQL in a Compute node (or using a plug-in node, since the Environment tree is available to all nodes) which sets a field thus:
    - SET Environment.Broker.Accounting.Origin = '...'
  - When the message has been processed by the flow the information identifying the origin is stored with all the other information for this origin and separate to information from other origins. Different origins can therefore be collected and reported separately.
- AccountingOrigin needs to be Enabled
  - This function is enabled using a new option on the mqsichangeflowstats command (see later). It is not always enabled because there is extra processing associated with this processing and although not excessive, it would have an impact on performance.
  - You should be aware that enabling this function could generate a large volume of reports, that's because you will get a different report for each AccountingOrigin identified by your flow in a given time period.
  - If the function is enabled, and **`Environment.Broker.Accounting.Origin`** is not populated then the Statistics and Accounting data collected while processing that input message will be accumulated to the default Accounting Origin.
    - The default value for the AccountingOrigin is "Anonymous".
  - If the Accounting Origin function is disabled then the default action is to accumulate all Accounting and Statistics data tothis default Origin.

Reduced Cost of Ownership

- **zSeries Application Assist Processor (zAAP) Exploitation**
  - Machine instructions generated by the Java Virtual Machine can be offloaded to dedicated processors called zAAPs .
  - Message Broker has several features which directly exploit zAAP technology, such as the Java Compute node, XLST node and JMS nodes .
  - Message transformations using the above nodes can offload the processing to zAAP's.
- **Getting Started Sub-Capacity Pricing (GSSP)**
- **Continuous performance improvements**

Notes : Reduced Cost of Ownership

- zAAP Exploitation.
  - Machine instructions generated by the Java Virtual Machine can be offloaded to dedicated processors called zAAPs.
  - zAAP costs significantly less than regular central processor
  - zAAP capacity is not included in MSU capacity
  - Java based applications can be off-loaded without increase in software costs.
  - Message Broker has several features which directly exploit zAAP technology, for example the Java Compute node, XLST node and JMS nodes.
  - It should be noted, however, parsing operations are still performed by non Java components and these are not eligible for offload.

z/OS Specific Nodes

- **QSAM Nodes (IA11)**
  - These are similar in concept and usage to the VSAM nodes, but oriented around sequential files, rather than record oriented files.
    - FileDelete
    - FileRead
    - FileRename
    - FileWrite
- **VSAM Nodes (IA13)**
  - A suite of nodes allowing users to perform record oriented processing on VSAM files.
    - VSAMDelete
    - VSAMInput
    - VSAMRead
    - VSAMUpdate
    - VSAMWrite

Notes : z/OS Specific Nodes

- VSAM Nodes
  - The broker can process VSAM records in the same way as it processes messages from or to other data sources.
  - A suite of 5 nodes allowing users to perform record oriented processing on VSAM files: Input, Read, Write, Update and Delete operations.
  - Users can combine these nodes for VSAM access with other nodes to integrate VSAM processing into message flows, or use VSAM files to drive message flow processing

- QSAM Nodes
  - These are similar in concept and usage to the VSAM nodes, but oriented around sequential files, rather than record oriented files.

Summary

- Runtime Environment for Brokering
- Functionally Complete and Consistent
- z/OS and z/Series Exploitation
- Extensive Collateral Technology
- Significant Version 7 Improvements
- Advanced and Mature Platform

The rest of the week ……

	Monday	Tuesday	Wednesday	Thursday	Friday
08:00			More than a buzzword: Extending the reach of your MQ messaging with Web 2.0	Batch, local, remote, and traditional MVS - file processing in Message Broker	Lyn's Story Time - Avoiding the MQ Problems Others have Hit
09:30		WebSphere MQ 101: Introduction to the world's leading messaging provider	The Do's and Don'ts of Queue Manager Performance	So, what else can I do? - MQ API beyond the basics	MQ Project Planning Session
11:00		MQ Publish/Subscribe	The Do's and Don'ts of Message Broker Performance	Diagnosing problems for Message Broker	What's new for the MQ Family and Message Broker
12:15	MQ Freebies! Top 5 SupportPacs	The doctor is in. Hands-on lab and lots of help with the MQ family		Using the WMQ V7 Verbs in CICS Programs	
01:30	Diagnosing problems for MQ	WebSphere Message Broker 101: The Swiss army knife for application integration	The Dark Side of Monitoring MQ - SMF 115 and 116 record reading and interpretation	Getting your MQ JMS applications running, with or without WAS	
03:00	Keeping your eye on it all - Queue Manager Monitoring & Auditing	The MQ API for dummies - the basics	Under the hood of Message Broker on z/OS - WLM, SMF and more	Message Broker Patterns - Generate applications in an instant	
04:30	Message Broker administration for dummies	All About WebSphere MQ File Transfer Edition	For your eyes only - WebSphere MQ Advanced Message Security	Keeping your MQ service up and running - Queue Manager clustering	
06:00			Free MQ! - MQ Clients and what you can do with them	MQ Q-Box - Open Microphone to ask the experts questions	

SHARE
Technology • Connections • Results