Service broker empowering real-time mainframe analytics

Selected Architecture and Techniques for success

Brendon Collier
Bill Fellows

dataview

About us…

- Bill Fellows is a SQL Server MVP, organizes the KC SQL Saturday, loves SSIS, Biml and long walks in the woods.
- Brendon Collier has worked as an SQL Server developer for the past 5 years, with a unhealthy focus on Service Broker. Prior to that he worked as a process improvement leader, manager and engineer. If he grows up he hopes to be a professional driver on a closed course.

dataview

What Data View offers

- Service Broker (obviously)
- Data Profiling and Analysis
- ETL and Business Intelligence Solutions
- ETL conversions (Informatica to SSIS)
- Mainframe Analytics

www.dataview-llc.com

Alfredo.Araiza@dataview-llc.com https://www.linkedin.com/in/alfredo-araiza-4b89952/

Brendon.Collier@dataview-llc.com https://www.linkedin.com/in/brendoncollier/

Bill.Fellows@dataview-llc.com https://www.linkedin.com/in/billinkc

Kevin.Mackey@dataview-llc.com https://www.linkedin.com/in/kevinjmackey/

dataview

What today's talk covers

- The abstract that made you sign up for this talk:
  - If you've learned the basics of Service Broker, this discussion will help you understand how to design a Service Broker architecture that allows for near real time message processing from external sources. In this case, see a system designed to handle **5 million varied messages** an hour in **less than 750 milliseconds per message** across multiple, distributed commodity SQL Servers. Discussion of service broker design and optimization, conversation management, error handling and logging, and operational metrics.
- A discussion of architecture for a Service Broker architecture of high speed / low latency
- Best practices we've learned for:
  - Overall Design
  - Conversation management
  - Message Batch Management
  - Activation Procedures
  - Queue / Service Design
  - Logging and metrics
  - Cross Server Broker use
- Some code samples

dataview

Prerequisite SQL Server & Broker concepts

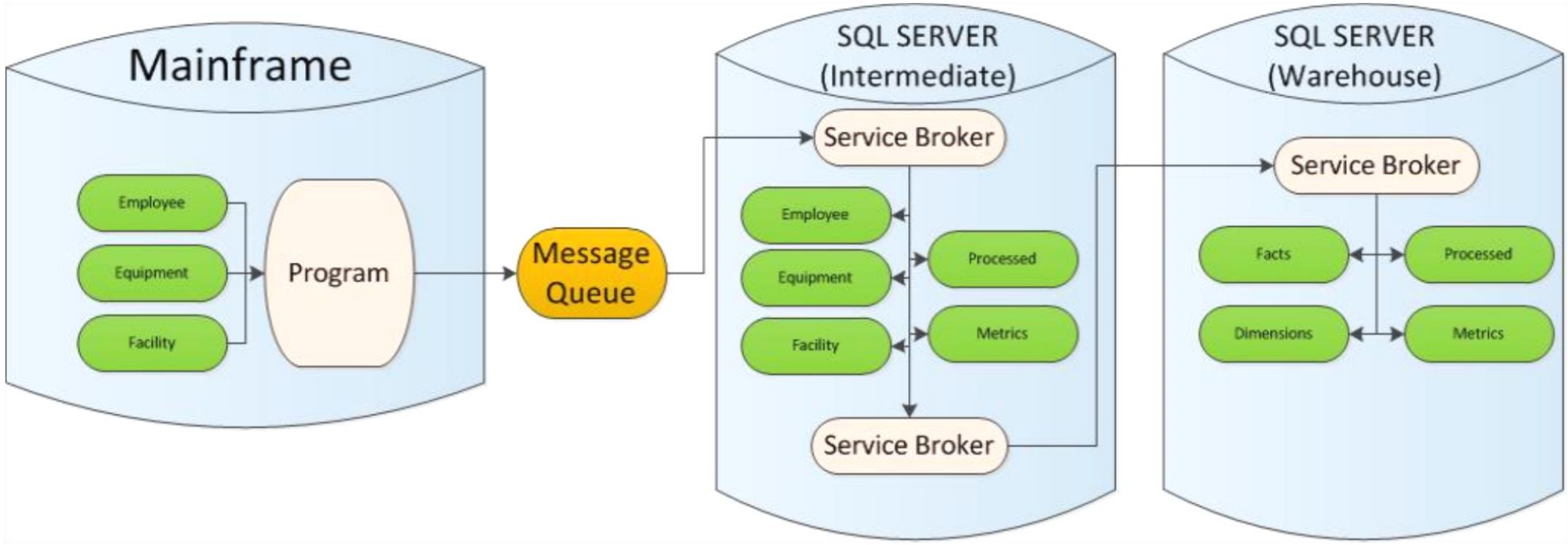
- Creation and maintenance of:
  - Messages
  - Queues
  - Services
  - Routes
  - Basic Activation Procedures
- And understand conceptually:
  - Broker Priority
  - Cross server broker use

Why do you need a Broker setup like this?

- You want to retire a legacy system in a staged manner
- You have legacy systems with limited BI capability, and you'd like BI data outside of batch runs
  - Legacy will remain, but the data needs to be more accessible
- You have modern COTS platforms which require data from legacy system as it occurs
  - Legacy remains and is accessible
- Inbound API messages which need to be translated and stored, especially in current SQL infrastructures

dataview

Data Flow Overview



dataview

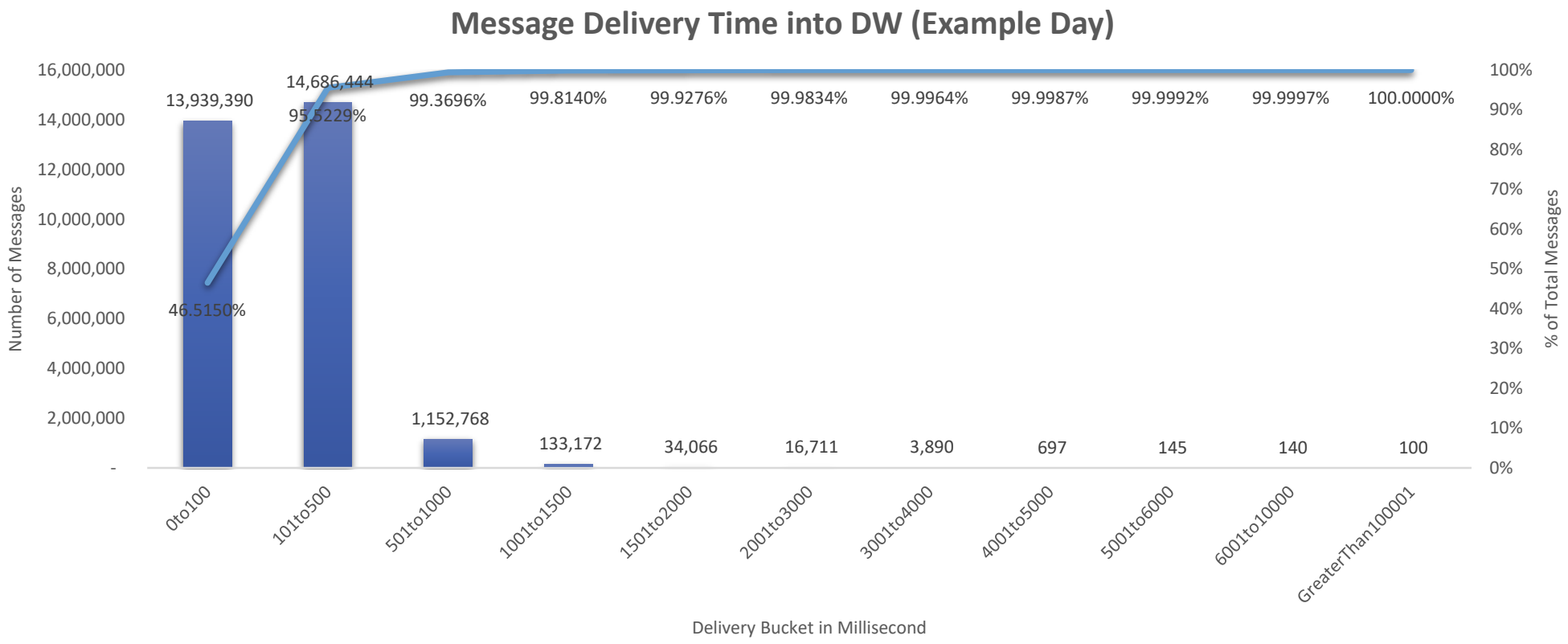
Terminology we'll use

- RTM = "real time messaging". It’s how we define the high speed message movement and the schema for those critical objects.
- Legacy system = The old school black and green source system you want to expose data from.
- Near Real Time = Data movement in under 1 second, from receipt to write
- Reoccur / Multiply Occurring Fields (MOF)
- Code Generation = Code generation is key to development speed
- Redgate - SQL Schema Compare / SQL Prompt = Generated code and broker code are nearly impossible without using! (a must for Database Developers)

dataview

What is high speed / volume, exactly?

- 29,967,523 total records over 143 queues (in DW. Higher in Intermediate)
- 320 tables written



dataview

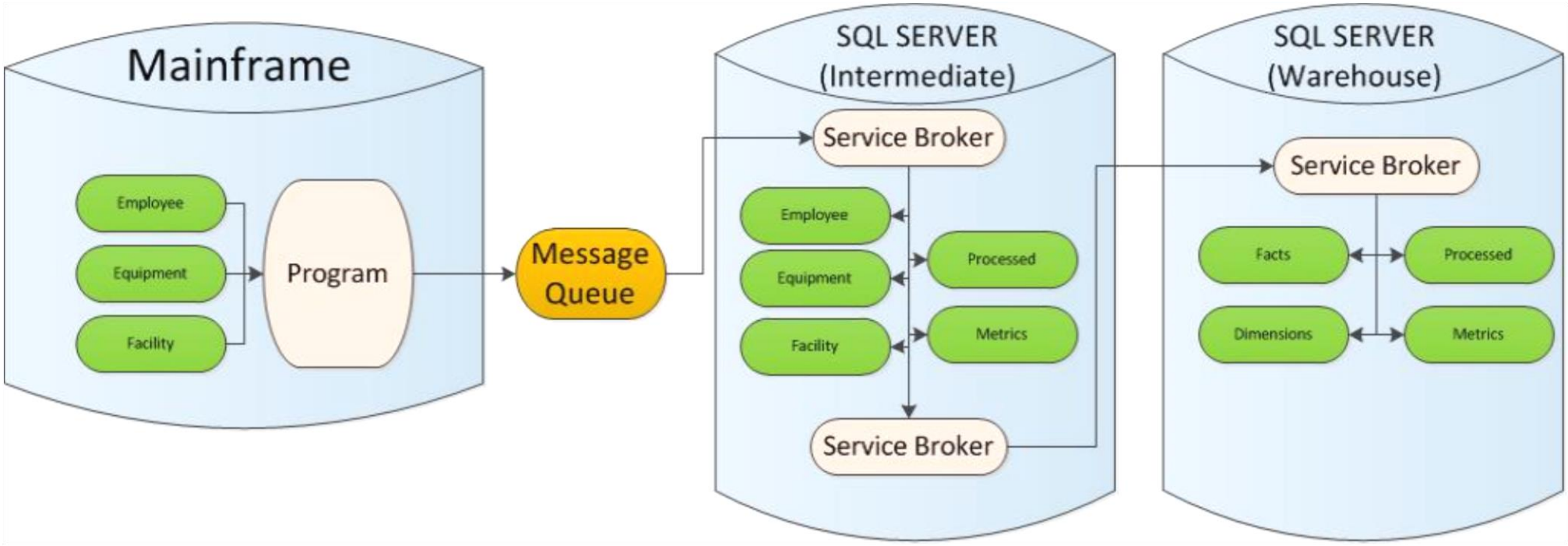
Overview of the high level architecture

- Using SQL Server 2012 thru 2017 (some modifications to suit version)
- JSON / XML enabled transactions on legacy system
- Messages dropped onto a queuing system in route to SQL/Broker
- To uniquely ID records, must have a unique key and modification time stamp and/or sequence maintained by source system (FILE, RECORD_KEY, DTS, Record_Sequence)
- Data is first persisted in an Intermediate Database (similar to a traditional Staging Database)
- Final destination is a typical Data Warehouse

dataview

Data Flow Overview

- Mainframe to External Message Queue to Intermediate and then to Warehouse



dataview

Key Architecture Components

- Message Resiliency (This is why we use Service Broker)
- Batch Processing
- Conditional Error Handling
- Table Driven Stored Procedure Configuration
- End Point Monitoring
- System Performance Metrics
- Data Validation

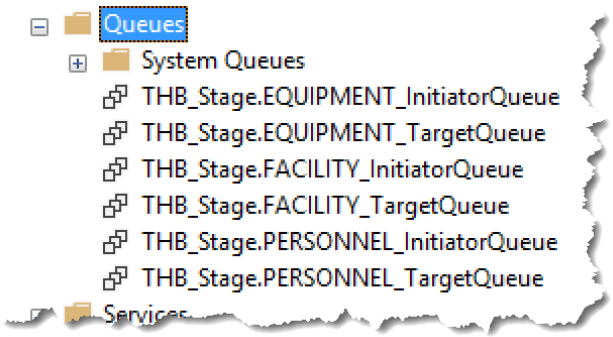
dataview

Getting from Legacy systems to Service Broker

- Queuing system that can call ODBC
  - Call `RTM.SendMessageToAnyService` and pass message
- Service Broker Message Types
  - XML, well formed not schema
  - JSON
  - XML or JSON Compressed as binary
  - Binary Messages
- All messages land in a Broker Queue for processing

dataview

Queue design



- Multiple Queues are preferred
  - May initially seem more complicated, but provides agility and simplifies maintenance operations
  - Be cautious with activation “max readers” – our experience is that 100 total queue readers are the max (32 core SQL Server)

```
SELECT * FROM [RTM].[ServiceBrokerStats_VW] [sbsv]
    WHERE [sbsv].[Queue] LIKE '%Target%'
```

90 %

Results | Messages

	Database_Name	schema	Queue	Queue_State	Messages_in_Queue	is_activation_enabled	current_readers	max_readers	last_empty_rowset_time	last_activated_time	queue_last_modify_date	tasks_wait
1	MFD_Code	THB_Stage	EQUIPMENT_TargetQueue	INACTIVE	2	1	NULL	4	2017-10-06 05:50:31.397	2017-10-06 22:09:36.263	2017-10-06 05:50:31.397	0
2	MFD_Code	THB_Stage	PERSONNEL_TargetQueue	INACTIVE	0	1	NULL	4	2017-10-06 05:50:36.307	2017-10-06 05:50:36.307	2017-10-06 05:50:36.307	0
3	MFD_Code	THB_Stage	FACILITY_TargetQueue	INACTIVE	0	1	NULL	4	2017-10-06 05:50:32.137	2017-10-06 05:50:32.137	2017-10-06 05:50:32.137	0

A queue per logical processing group

Current total readers is important to overall system health

dataview

Queue Design (2)

Multiple services can be attached to queue and used for message receipt

MFD_FACILITY_TargetService
MFD_FACILITY_XML_TargetService

THB_Stage.FACILITY_InitiatorQueue
THB_Stage.FACILITY_TargetQueue

Target queue is most important

Nearly always ON. Turning OFF means messages are not able to enter queue

OFF: A very sneaky setting… messages will persist and you'll scream "why??"

ON or OFF. This is the setting to hold messages in a queue

```
ALTER QUEUE [THB_Stage].[PERSONNEL_TargetQueue] WITH STATUS = ON , RETENTION = OFF , ACTIVATION (  STATUS = ON , PROCEDURE_NAME = [THB_Stage].
  [PERSONNEL_Receive_XML_Message] , MAX_QUEUE_READERS = 4 , EXECUTE AS N'ServiceBroker'  ), POISON_MESSAGE_HANDLING (STATUS = OFF)
```

OFF: Messages should not be rolled back to queue. Also a potential silent failure point if queue is inactivated

dataview

Conversation management

- All Messages require a conversation to enter the queue
- Conversation management revolves around `sys.conversation_endpoints` and `SEND ON CONVERSATION`
- RTM "enhancements" help manage

dataview

Downsides of a basic broker send

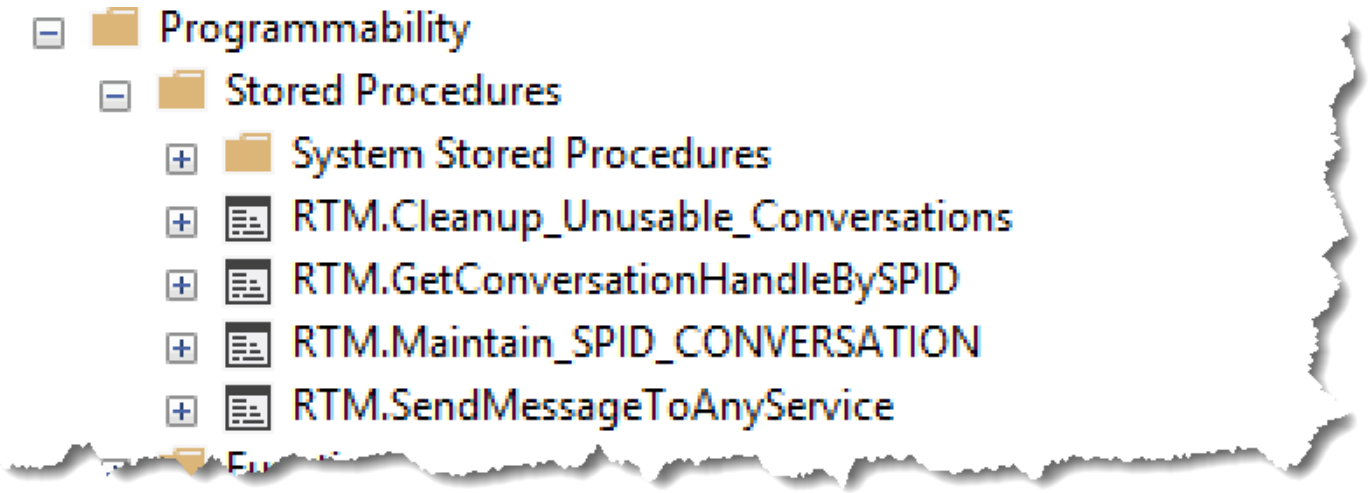
```sql
/*traditional send via broker*/

DECLARE @MessageBody XML = ( SELECT 'Message' =  'Foo' FOR XML PATH)
DECLARE @ch UNIQUEIDENTIFIER
--first get a conversation

    BEGIN DIALOG CONVERSATION @ch
    FROM SERVICE EQUIPMENT_InitiatorService
    TO SERVICE 'MFD_EQUIPMENT_TargetService'
    ON CONTRACT XML_Message_Contract
    WITH ENCRYPTION = OFF;

--send message on that conversation

SEND ON CONVERSATION @ch MESSAGE TYPE XML_Message (@MessageBody);

-- end conversation
END CONVERSATION @ch
```

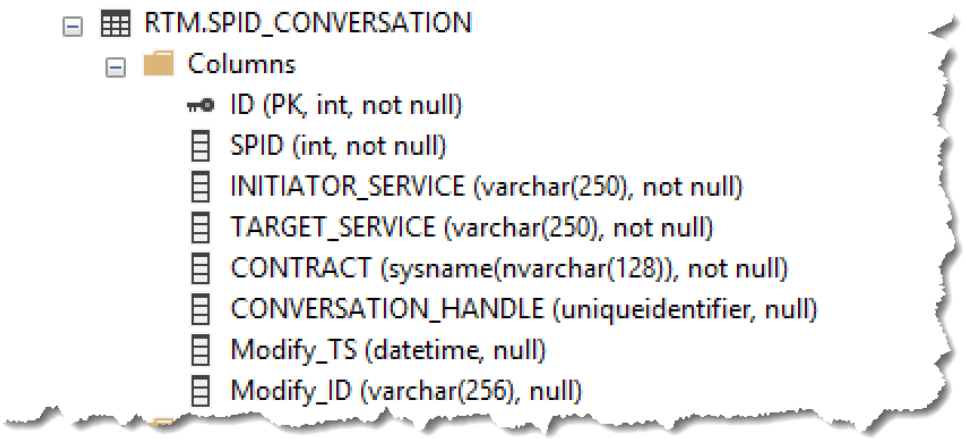
- Cost of creating a new conversation with each send
- “Send ON” has atomic transaction management… XACT_ABORT status is important for managing failures correctly
- In a high volume environment, a high probability of deadlocks on `sys.conversation_endpoints` during conversation creation and retrieval exists
- No consistency on conversation `LIFETIME` across developer use

dataview

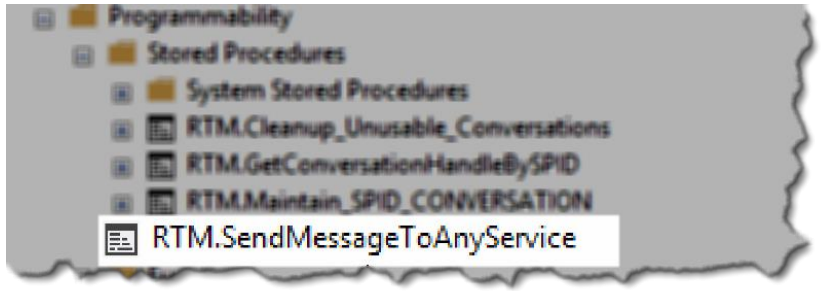
A better way to send…

- Set of Stored Procedures, along with a single table, to handle Sending and Conversation Management





A better way to send… (2)

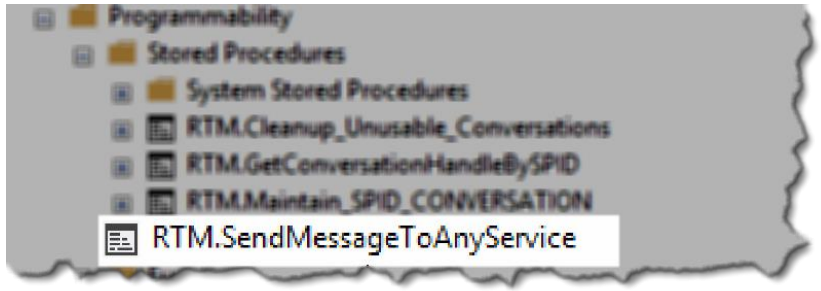



```
ALTER PROCEDURE RTM.SendMessageToAnyService
    IF ( ( 16384 & @@OPTIONS ) = 16384 )
    TRY
        IF @TransactionCount = 0
            BEGIN TRANSACTION
        EXECUTE RTM.GetConversationHandleBySPID
        IF ( @ch IS NULL )
            BEGIN DIALOG CONVERSATION
        IF @TransactionCount = 0
            COMMIT TRANSACTION
    CATCH
        IF ( Xact_State() ) = -1
            IF @TransactionCount = 0
                ROLLBACK TRANSACTION
            ELSE
                ROLLBACK TRANSACTION
            EXECUTE RTM.AddEvent
        IF ( Xact_State() ) = 1 AND @TransactionCount = 0
            ROLLBACK TRANSACTION
            EXECUTE RTM.AddEvent
        IF ( Xact_State() ) = 1 AND @TransactionCount > 0
            ROLLBACK TRANSACTION
            EXECUTE RTM.AddEvent
        INSERT INTO RTM.Process_M204_ErrorLog
    TRY
        SEND ON CONVERSATION @ch
        IF @GetCHSuccess = 0
            END CONVERSATION @ch
    CATCH
        EXECUTE RTM.AddEvent
        INSERT INTO RTM.Process_M204_ErrorLog
    IF @XACT_ABORT = 1
```

```
ALTER PROCEDURE [RTM].[SendMessageToAnyService] (
    @FromService sysname
   ,@ToService sysname
   ,@Contract sysname
   ,@MessageType sysname
   ,@MessageBody XML )
AS
```

dataview

A better way to send… (3)



- Transaction management is critical around “send”
- The SP is relatively basic, however the transaction management is carefully calculated

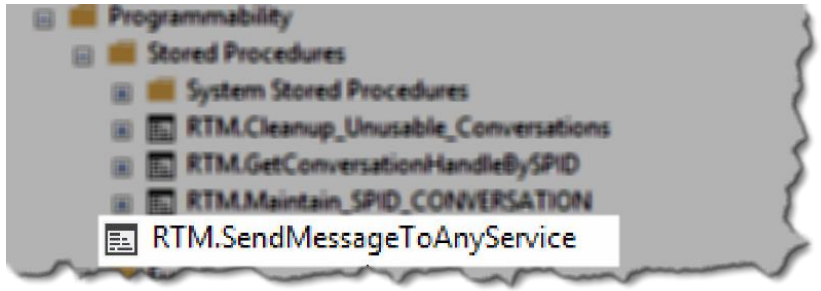
```
--Find the starting state of XACT_ABORT
DECLARE @XACT_ABORT BIT = 0;
IF (( 16384 & @@OPTIONS ) = 16384 )
    SET @XACT_ABORT = 1;

--turn XACT_ABORT off to prevent transactions from rolling back automatically
SET XACT_ABORT OFF;

--prior to SP end
IF @XACT_ABORT = 1
    BEGIN
        SET XACT_ABORT ON;
    END;
```

```
--obtain the latest open conversation handle
    EXECUTE [RTM].[GetConversationHandleBySPID]
        @InitiatorService = @FromService
       ,@TargetService = @ToService
       ,@Contract = @Contract
       ,@ch = @ch OUTPUT;
```

```
SEND ON CONVERSATION @ch MESSAGE TYPE @MessageType (@MessageBody);
```

dataview

A better way to send… (4)



- Common transaction management in catch block to roll back to caller correctly – In **almost ALL of our RTM code**
- Evaluate the transaction count when called
- `SET @TransactionCount = @@TRANCOUNT;`
- Rollback correctly

```
BEGIN
    IF ( Xact_State() ) = -1
        BEGIN
            IF @TransactionCount = 0
                BEGIN
                    ROLLBACK TRANSACTION;
                    SET @details = N'The transaction is in an uncommittable state. Rolling back transaction.'
                END

            ELSE
                BEGIN
                    ROLLBACK TRANSACTION [SavePoint]
                    SET @details = N'The transaction is in an uncommittable state. Rolling back transaction to Save Point.'
                END

            EXEC [RTM].[AddEvent]
                @Level = N'WARNING'
               ,@Source = N'[RTM].[SendMessageToAnyService]'
               ,@Details = @details
               ,@IntResult = 0
               ,@SessionId = NULL
               ,@Message_Body = @xml_error_details;
        END;

    IF ( Xact_State() ) = 1 AND
        @TransactionCount = 0
        BEGIN
            ROLLBACK TRANSACTION;
            EXEC [RTM].[AddEvent]...;
        END;

    IF ( Xact_State() ) = 1 AND
        @TransactionCount > 0
        BEGIN
            ROLLBACK TRANSACTION [SavePoint];
            EXEC [RTM].[AddEvent]...;
        END;
END
```

dataview

A better way to send… (6)

- Focus on reducing reads and deadlock potential on `sys.conversation_endpoints`

```
--set lifespan in seconds (24 hours x 60 minutes x 60 seconds) X 4 days
    DECLARE @lifetime INT = ( 24 * 60 * 60 ) * 4;
    DECLARE @SessionID UNIQUEIDENTIFIER = NewId();
    DECLARE @RandomExecutionRate INT = 20000
    DECLARE @Expiration_Minutes_delta INT = 30
    DECLARE @Expiration_delta DATETIME = DateAdd(MINUTE, @Expiration_Minutes_delta, GetUtcDate())
```

Lifetimes are managed and relatively short

To avoid the need for jobs and be responsive to message volume, we execute maintenance SP on an interval

```
IF (
      SELECT (Round ((Rand () * (@RandomExecutionRate - 1)), 0) + 1)
    ) = 1
    BEGIN TRY
        BEGIN
            EXEC [RTM].[Maintain_SPID_CONVERSATION]
                @SessionID;
        END;
```

dataview

A better way to send… (7)

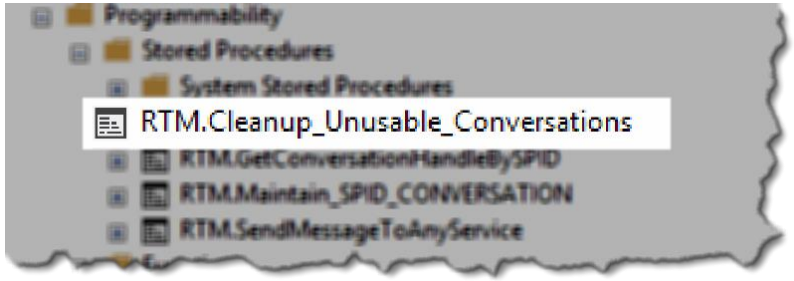
The things that make a unique conversation are kept by SPID, which provides individual conversations for each process

New conversations are logged into table. Conversation handle is returned to caller.

```
--Find a ch that is valid for at least N expiration minutes and return as @ch
    SELECT TOP 1
            @ch = [ce].[conversation_handle]
        FROM
            [RTM].[SPID_CONVERSATION] [sc] WITH ( NOLOCK )
        JOIN [sys].[conversation_endpoints] [ce] WITH ( NOLOCK ) ON [sc].[CONVERSATION_HANDLE] = [ce].[conversation_handle] AND
                                                                    [ce].[state] IN ( 'CO', 'SO' )
        WHERE
            [sc].[SPID] = @@SPID AND
            [sc].[INITIATOR_SERVICE] = @InitiatorService AND
            [sc].[TARGET_SERVICE] = @TargetService AND
            [sc].[CONTRACT] = @Contract AND
            [ce].[lifetime] > @Expiration_delta

    --if we don't find a ch, make one
    IF @ch IS NULL
        BEGIN
            BEGIN DIALOG CONVERSATION @ch
            FROM SERVICE @InitiatorService
            TO SERVICE @TargetService
            ON CONTRACT @Contract
            WITH ENCRYPTION = OFF, LIFETIME = @lifetime;

            INSERT [RTM].[SPID_CONVERSATION]
                   ( [SPID]
                   ,[INITIATOR_SERVICE]
```

dataview

A better way to send… (9)




```
ALTER PROCEDURE [RTM].[Cleanup_Unusable_Conversations] (
    @SessionID UNIQUEIDENTIFIER = NULL )
AS
    BEGIN

        SET LOCK_TIMEOUT 2000;

        DECLARE @cleanup_lag_minutes INT = 360 --6 hours
        DECLARE @cleanup_lag_TS DATETIME = DateAdd(MINUTE, @cleanup_lag_minutes, GetUtcDate())
```

Explicit lock timeout, again

Set a time threshold to hold conversations after expiration

```
INSERT INTO @convolist
        SELECT TOP 500
                [ce].[conversation_handle]
               ,[ce].[far_service]
               ,[ce].[state]
               ,[ce].[receive_sequence]
               ,[ce].[send_sequence]
               ,[ce].[lifetime]
            FROM
                [sys].[conversation_endpoints] [ce] WITH ( READPAST )
            WHERE
                [ce].[state] IN ( 'ER', 'DI', 'CD', 'DO' ) AND
                GetUtcDate() >= DateAdd(MINUTE, @cleanup_lag_minutes, [ce].[lifetime]) AND
                NOT EXISTS ( SELECT
                                1
                            FROM
                                [sys].[transmission_queue] AS [TQ] WITH ( NOLOCK )
                            WHERE
                                [ce].[conversation_handle] = [TQ].[conversation_handle] );
```

Find those conversations that cannot be used and are not ended, then cursor thru and end them

Do NOT risk message loss: Check sys.transmission_queue for messages in a given conversation

A better way to send… (10)

- Maintenance on SPID_Conversation table is required as conversations end
- Maintenance job is called by GetConversationHandle
- We work very hard to make sure maintenance job is low impact on speed and latency

```
ALTER PROCEDURE  RTM.Maintain_SPID_CONVERSATION
    TRY
        IF  @TransactionCount = 0
            BEGIN TRANSACTION
        DELETE FROM  sc
        IF  @SessionID IS NULL
        IF  @DeleteCount > 0
        EXECUTE  RTM.AddEvent
        IF  @TransactionCount = 0
            COMMIT TRANSACTION
    CATCH
        IF  ( XACT_STATE() ) = -1
        IF  ( XACT_STATE() ) = 1 AND @TransactionCount = 0
        IF  ( XACT_STATE() ) = 1 AND @TransactionCount > 0
```

dataview

Once you've sent a message, you need to do something: Activation Procedures

Activation Procedures goals

2 major SP patterns

- Intermediate database
  - Read from queue and parse message
  - Normalizing Reoccurring fields
  - Writing untyped data to limited tables
  - Sending messages to DW and other targets
- Warehouse
  - Read from queue and parse message
  - Sorting record types
  - Typing data and testing for type compliance
  - Writing typed data numerous tables

Common Design Patterns within

1. External Configuration Store Pattern
2. Runtime Reconfiguration Pattern
3. JSON or XML parsing using CLR
4. Looping
5. Batch management
6. Retry Pattern
7. Circuit Breaker Pattern
8. Health Endpoint Monitoring Pattern
9. Sharding Pattern
10. In memory use

dataview

Activation Procedures, High Level

- Activation SP's commit the transaction from the queue :: We **DO NOT rollback to the queue** in case of error!!
- The transaction is
  1. Written to intended target(s)
  2. Written to error table
  3. Retried in proc
  4. Retried by sending back to queue (a new send call)

dataview

Why we do not rollback to queues

- Batched XML contains multiple source messages, which must be handled individually
- In this case, record 3 within the batch fails, however the other 4 records are valid.
- Rolling back to the queue does not correct the error, and does not handle the 4 valid records


```
<row>
  <id>1</id>
  <RECORD_KEY>100</RECORD_KEY>
  <IntData>2</IntData>
  <Sent_TS>2017-10-06T20:36:02.4274688</Sent_TS>
</row>
<row>
  <id>2</id>
  <RECORD_KEY>200</RECORD_KEY>
  <IntData>5</IntData>
  <Sent_TS>2017-10-06T20:36:02.4274688</Sent_TS>
<row>
  <id>3</id>
  <RECORD_KEY>300</RECORD_KEY>
  <IntData>Bad Data</IntData>
  <Sent_TS>2017-10-06T20:36:02.4274688</Sent_TS>
</row>
  <RECORD_KEY>100</RECORD_KEY>
  <IntData>6</IntData>
  <Sent_TS>2017-10-06T20:36:02.4274688</Sent_TS>
</row>
<row>
  <id>5</id>
  <RECORD_KEY>500</RECORD_KEY>
  <IntData>9</IntData>
  <Sent_TS>2017-10-06T20:36:02.4274688</Sent_TS>
</row>
```


dataview

Batches of batches, oh my!

```sql
SELECT casted_message_body = CAST(message_body AS XML) , conversation_handle, service_name, message_enqueue_time
FROM   [MFD_Code].[THB_Stage].[EQUIPMENT_TargetQueue] WITH ( NOLOCK );
```

0 %

Results | Messages

	casted_message_body	conversation_handle	service_name	message_enqueue_time
1	<row><id>1</id><RECORD_KEY>100</RECORD_KEY><IntData>2</IntData><Sent_TS>2017-10-06T20:3...	83C871F7-D5AA-E711-A94C-000D3A93CE6F	MFD_EQUIPMENT_TargetService	2017-10-06 20:36:02.437
2	<row><id>6</id><RECORD_KEY>900</RECORD_KEY><IntData>0</IntData><Sent_TS>2017-10-06T20:3...	83C871F7-D5AA-E711-A94C-000D3A93CE6F	MFD_EQUIPMENT_TargetService	2017-10-06 20:36:02.440

```xml
<row>
  <id>1</id>
  <RECORD_KEY>100</RECORD_KEY>
  <IntData>2</IntData>
  <Sent_TS>2017-10-06T20:36:02.4274688</Sent_TS>
</row>
<row>
  <id>2</id>
  <RECORD_KEY>200</RECORD_KEY>
  <IntData>5</IntData>
  <Sent_TS>2017-10-06T20:36:02.4274688</Sent_TS>
</row>
<row>
  <id>3</id>
  <RECORD_KEY>300</RECORD_KEY>
  <IntData>Bad Data</IntData>
  <Sent_TS>2017-10-06T20:36:02.4274688</Sent_TS>
</row>
<row>
  <id>4</id>
  <RECORD_KEY>100</RECORD_KEY>
  <IntData>6</IntData>
  <Sent_TS>2017-10-06T20:36:02.4274688</Sent_TS>
</row>
<row>
  <id>5</id>
  <RECORD_KEY>500</RECORD_KEY>
  <IntData>9</IntData>
  <Sent_TS>2017-10-06T20:36:02.4274688</Sent_TS>
</row>
```

Broker Message 1 contains 5 messages in XML. Record 3 will fail data typing

```xml
<row>
  <id>6</id>
  <RECORD_KEY>900</RECORD_KEY>
  <IntData>0</IntData>
  <Sent_TS>2017-10-06T20:36:02.4274688</Sent_TS>
</row>
<row>
  <id>7</id>
  <RECORD_KEY>700</RECORD_KEY>
  <IntData>45</IntData>
  <Sent_TS>2017-10-06T20:36:02.4274688</Sent_TS>
</row>
<row>
  <id>8</id>
  <RECORD_KEY>800</RECORD_KEY>
  <IntData>11</IntData>
  <Sent_TS>2017-10-06T20:36:02.4274688</Sent_TS>
</row>
```

Broker Message 2 contains 3 messages in XML. All pass data typing

dataview

Activation SP :: What's in the XML Message


```xml
<message>
  <dataInput>
    <hdr>
      <programId>EQUIP.100</programId>
      <user>BATCH1</user>
    </hdr>
    <payload>
      <control>
        <dts>3667420681456</dts>
        <fgname>EQUIPMENT</fgname>
        <tranid>0429536674206814 56</tranid>
      </control>
      <record>
        <RECORD.KEY>20160317172802931003168140 2339</RECORD.KEY>
        <DTS>20160319235801456</DTS>
        <EQUIPMENT.NUMBER>1458</EQUIPMENT.NUMBER>
        <EQUIPMENT.WEIGHT>1500</EQUIPMENT.WEIGHT>
        <EQUIPMENT.LOCATION>OPKS</EQUIPMENT.LOCATION>
        <EQUIPMENT.DIMENSION>100</EQUIPMENT.DIMENSION>
        <EQUIPMENT.DIMENSION>250</EQUIPMENT.DIMENSION>
        <EQUIPMENT.DIMENSION>45</EQUIPMENT.DIMENSION>
      </record>
    </payload>
  </dataInput>
</message>
```


MOF Example


```xml
<record>
  <RECORD.KEY>20160317172802931003168140 2339</RECORD.KEY>
  <DTS>20160319235801456</DTS>
  <EQUIPMENT.NUMBER>1458</EQUIPMENT.NUMBER>
  <EQUIPMENT.WEIGHT>1500</EQUIPMENT.WEIGHT>
  <EQUIPMENT.LOCATION>OPKS</EQUIPMENT.LOCATION>
  <EQUIPMENT.DIMENSION>100</EQUIPMENT.DIMENSION>
  <EQUIPMENT.DIMENSION>250</EQUIPMENT.DIMENSION>
  <EQUIPMENT.DIMENSION>45</EQUIPMENT.DIMENSION>
</record>
```


- When a Mainframe program modifies a record an XML Message is generated.
- Each message is a representation of the entire record within Mainframe at that point in time
- The key fields for each message are "Record.Key" and "DTS"
- Record.Key is the unique identifier for each record
- DTS. is the timestamp when the record was modified and the XML Message generated

dataview

Activation SP :: CLR Parsing of XML

- The messages are now are parsed using C# and .NET XmlReader. The biggest advantage over TSQL is that we don’t have to make multiple passes over the XML message when dealing with Multiply Occurring Fields (MOFs)

```
            while (reader.Read())
            {
                //Only looking for the start of elements
                if (reader.IsStartElement()|| reader.MoveToContent() == XmlNodeType.Element)
                {
                    //Get element name and switch on it
                    switch (reader.Name)
                    {
                        case "action":
                            baseRecord.SOURCEDELETED = (reader.ReadString() == "DELETE" ? true : false);
                            break;
                        case "programId":
                            baseRecord.programId = reader.ReadString();
                            break;
                        case "dateTime":
                            baseRecord.dateTime = reader.ReadString();
                            break;
                        case "tranid":
                            baseRecord.tranid = SqlInt64.Parse(reader.ReadString());
                            break;
                        case "RECORD.KEY":
                            baseRecord.RECORD_KEY = reader.ReadString();
                            break;
                        case "DTS.DTTMSP":
                            baseRecord.DTS_DTTMSP = SqlInt64.Parse(reader.ReadString());
                            break;
                        case "CAR.REASON.CD":
                            baseRecord.CAR_REASON_CD = reader.ReadString();
                            break;
                        case "CAR.REASON.REF.NBR":
                            baseRecord.CAR_REASON_REF_NBR = reader.ReadString();
                            break;
                        case "CAR.WRKGRP":
                            baseRecord.CAR_WRKGRP = reader.ReadString();
                            break;
                        case "CARE.SHP.TERMS":
                            baseRecord.CARE_SHP_TERMS = reader.ReadString();
                            break;
                        case "CARS.AUDIT.ACTION":
                            baseRecord.CARS_AUDIT_ACTION = reader.ReadString();
                            break;
                        case "CARS.CONSG.ATTN.LINE":
                            if(reader.GetAttribute("encoding")=="base64")
                            {
                                string value = Converter.TranslateB64EbcdicToUTF8(reader.ReadString());
                                if (value.Length<=255)
                                {
                                    baseRecord.CARS_CONSG_ATTN_LINE = value;
                                }
                                else
                                {
                                    baseRecord.CARS_CONSG_ATTN_LINE = value.Substring(0, 255);
                                }
                                baseRecord.encoded = true;
                            }
                            else
                            {
                                baseRecord.CARS_CONSG_ATTN_LINE = reader.ReadString();
                            }
```


dataview

Activation SP :: External Configuration

- RTM.ProcedureConfiguration
  - Columns
    - id (int, not null)
    - SchemaName (PK, varchar(255), not null)
    - ProcedureName (PK, varchar(255), not null)
    - varName (PK, varchar(255), not null)
    - varValueInt (int, null)
    - varValueString (varchar(
    - description (varchar(255)
    - Modify_TS (datetime2(7)
    - Modify_ID (varchar(256),

```sql
SELECT  [ProcedureName]
       ,[varName]
       ,[varValueInt]
       ,[varValueString]
       ,[description]
FROM    [RTM].[ProcedureConfiguration] WITH ( NOLOCK
WHERE   [SchemaName] = OBJECT_SCHEMA_NAME(@@PROCID)
        AND [ProcedureName] = OBJECT_NAME(@@PROCID);

SELECT  @LoopCounterMax = [varValueInt]
FROM    @v
WHERE   [varName] = 'LoopCounterMax';
SELECT  @ReceiveBatchSize = [varValueInt]
FROM    @v
WHERE   [varName] = 'ReceiveBatchSize';
SELECT  @ReceiveTimeout = [varValueInt]
FROM    @v
WHERE   [varName] = 'ReceiveTimeout';
SELECT  @MaxRetry = [varValueInt]
FROM    @v
WHERE   [varName] = 'DBWriteRetry';
SELECT  @SendToMetrics = [varValueInt]
FROM    @v
WHERE   [varName] = 'SendToMetrics';
SELECT  @SendToAppQueue = [varValueInt]
FROM    @v
WHERE   [varName] = 'SendToAppQueue';
SELECT  @ActivationProcedureDelay = [varValueString]
FROM    @v
```

- Read values from Procedure Configuration Table to dynamically control the Activation SP logic
  - @MaxLoopCounter
  - @MaxRetry
  - @WriteToProcessedTable
  - @SendFullXMLToWarehouse
- Used to control a number of activation procedure parameters dynamically
  - When In Memory is available, select directly from table
  - From disk table, insert into table variable first, then access

dataview

Activation SP :: While Loop

Why use it?

- It is more efficient to execute a stored procedure multiple times vs. a single activation every time you pull messages from the queue
- It can take broker up to 5 seconds to activate a stored procedure once a message hits a queue
- Coupled with the @MaxLoopCounter variable you can gracefully change the execution behavior of a proc

```
SELECT FROM  @v
WHILE  @LoopCounter < @MaxLoopCounter
    BEGIN TRANSACTION
    TRY
        RECEIVE FROM  EQP_TargetQueue
        IF  @@ROWCOUNT = 0
        COMMIT TRANSACTION
    CATCH
    IF  ( SELECT COUNT(1) FROM @tableMessages ) >= 1
        TRY
            IF  ( SELECT COUNT(1) FROM @MOF ) >= 1
            IF  EXISTS (SELECT COUNT(1) FROM @Messages WHERE [ProcessFlag] = 1)
                UPDATE  M
                WHILE  @InsertRetryCount < @MaxRetry AND @InsertSuccess = 0
                    TRY
                        BEGIN TRANSACTION
                        INSERT INTO  Legacy.Mainframe_EQP
                        COMMIT TRANSACTION
                    CATCH
                WHILE  @UpdateRetryCount < @MaxRetry AND @UpdateSuccess = 0
                    TRY
                        BEGIN TRANSACTION
                        UPDATE  T
                        COMMIT TRANS
```

dataview

Activation SP :: Processed Table

- Keep a log of all messages received from the source system
- Stores unparsed XML messages
- Extremely useful for troubleshooting and re-processing of messages

```
WHILE @DeleteRetryCount < @MaxRetry AND @DeleteSuccess = 0
    TRY
        BEGIN TRANSACTION
        DELETE FROM T
        COMMIT TRANSACTION
    CATCH
WHILE @InsertRetryCount < @MaxRetry AND @InsertSuccess = 0
    TRY
        BEGIN TRANSACTION
        INSERT INTO Legacy.Mainframe_EQP_MOF
        COMMIT TRANSACTION
    CATCH
IF @WriteToProcessedTable = 1
    INSERT INTO Legacy.Mainframe_EQP_Processed
IF @SendToMetrics = 1
    TRY
        INSERT INTO RTM.ServiceBroker_Metrics_EQP
    CATCH
IF @SendFullXMLToWarehouse = 1
    IF @xmlToWarehouse IS NOT NULL
        TRY
            BEGIN TRANSACTION
            EXECUTE RTM.SendMessageToAnyService
            COMMIT TRANSACTION
        CATCH
CATCH
```

dataview

Processed Table

- For troubleshooting purposes, all XML Messages received are stored in a “Processed” table for a period of up to 24 hours
- The sample below illustrates the “chattiness” of the Mainframe programs. In the span of 149 milliseconds 11 XML Messages were generated for a single record. SQL Server received all 11 records at the same time. Note the entire batch was processed in 101 milliseconds.

RECORD_KEY	DTS	ProgramId	ProcessFlag	INSERT_TS	ExecutionID	payload
20171005083858407063334630402 3	20171005083903924	Legacy	1	7:39:04 AM	6DD73A95-9397-4E1F-985C-1C64756F6449	<message><dataInput><hdr><programId>
20171005083858407063334630402 3	20171005083903920	Legacy	2	7:39:04 AM	6DD73A95-9397-4E1F-985C-1C64756F6449	<message><dataInput><hdr><programId>
20171005083858407063334630402 3	20171005083903916	Legacy	3	7:39:04 AM	6DD73A95-9397-4E1F-985C-1C64756F6449	<message><dataInput><hdr><programId>
20171005083858407063334630402 3	20171005083903904	Legacy	4	7:39:04 AM	6DD73A95-9397-4E1F-985C-1C64756F6449	<message><dataInput><hdr><programId>
20171005083858407063334630402 3	20171005083903900	Legacy	5	7:39:04 AM	6DD73A95-9397-4E1F-985C-1C64756F6449	<message><dataInput><hdr><programId>
20171005083858407063334630402 3	20171005083903872	Legacy	6	7:39:04 AM	6DD73A95-9397-4E1F-985C-1C64756F6449	<message><dataInput><hdr><programId>
20171005083858407063334630402 3	20171005083903862	Legacy	7	7:39:04 AM	6DD73A95-9397-4E1F-985C-1C64756F6449	<message><dataInput><hdr><programId>
20171005083858407063334630402 3	20171005083903807	Legacy	8	7:39:04 AM	6DD73A95-9397-4E1F-985C-1C64756F6449	<message><dataInput><hdr><programId>
20171005083858407063334630402 3	20171005083903786	Legacy	9	7:39:04 AM	6DD73A95-9397-4E1F-985C-1C64756F6449	<message><dataInput><hdr><programId>
20171005083858407063334630402 3	20171005083903781	Legacy	10	7:39:04 AM	6DD73A95-9397-4E1F-985C-1C64756F6449	<message><dataInput><hdr><programId>
20171005083858407063334630402 3	20171005083903775	Legacy	11	7:39:04 AM	6DD73A95-9397-4E1F-985C-1C64756F6449	<message><dataInput><hdr><programId>

Queue	Delivery_Lag_MS	Processing_Time_MS	Received_Count	Processed_Count	Insert_TS	ExecutionID
Equipment	97	101	26	4	2017-10-05 07:39:04.030	6dd73a95-9397-4e1f-985c-1c64756f6449

Activation SP :: Error Handling and Logging

- Wrap all transactions with a TRY/Catch
- Log all errors to an Error table for analysis and re-processing

```
SELECT FROM  @v
WHILE  @LoopCounter < @MaxLoopCounter
  BEGIN TRANSACTION
  TRY
    RECEIVE FROM  EQP_TargetQueue
    IF  @@ROWCOUNT = 0
    COMMIT TRANSACTION
  CATCH
  IF  ( SELECT COUNT(1) FROM @tableMessages ) >= 1
    TRY
      IF  ( SELECT COUNT(1) FROM @MOF ) >= 1
      IF  EXISTS (SELECT COUNT(1) FROM @Messages WHERE [ProcessFlag] = 1)
        UPDATE  M
        WHILE  @InsertRetryCount < @MaxRetry AND @InsertSuccess = 0
          TRY
            BEGIN TRANSACTION
            INSERT INTO  Legacy.Mainframe_EQP
            COMMIT TRANSACTION
          CATCH
        WHILE  @UpdateRetryCount < @MaxRetry AND @UpdateSuccess = 0
          TRY
            BEGIN TRANSACTION
            UPDATE  T
            COMMIT TRANS
```

dataview

Activation SP :: Metrics Logging

To monitor the overall health of the environment you should log performance metrics

- Delivery Time from Source System
- Processing Time
- Message Volume
- Received vs. Processed Count
- Any other relevant metrics which can be used to gauge performance

```
WHILE @DeleteRetryCount < @MaxRetry AND @DeleteSuccess = 0
    TRY
        BEGIN TRANSACTION
        DELETE FROM T
        COMMIT TRANSACTION
    CATCH
WHILE @InsertRetryCount < @MaxRetry AND @InsertSuccess = 0
    TRY
        BEGIN TRANSACTION
        INSERT INTO Legacy.Mainframe_EQP_MOF
        COMMIT TRANSACTION
    CATCH
IF @WriteToProcessedTable = 1
    INSERT INTO Legacy.Mainframe_EQP_Processed
IF @SendToMetrics = 1
    TRY
        INSERT INTO RTM.ServiceBroker_Metrics_EQP
    CATCH
IF @SendFullXMLToWarehouse = 1
    IF @xmlToWarehouse IS NOT NULL
        TRY
            BEGIN TRANSACTION
            EXECUTE RTM.SendMessageToAnyService
            COMMIT TRANSACTION
        CATCH
CATCH
```

dataview

Endpoint Monitoring

SSRS Dashboard is used to monitor the following:

- Message delivery and processing time
- Total messages received
- Broker Transmission Queue Status
- Active Broker Queue Reader Count and Status
- Number of messages batches pending processing for each queue

Transmission Queue Status

Message Count	Alert Level
571	INFO

Currently Active Queue Readers

Current Reader Count
22

Error Log

No Errors Found to Report

ADJ	ASH	ATC	ATT	ATV	BAC	BATCHMSG	BMF	BOA	BRBRSC	BRCRSC	BSP	BSPAUD	BSR	BSRPDB	BSRPRE	BSS
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
BUS	BUSAUDIT	BUSCMT	CAR	CARE	CARH	CARS	CCD	CCLS	CCS	CDT	CIF	CINV	CIVR	CLF	CLFTEC	CLOC
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CMS	CMT	CNT	COLA	COLC	COLE	COLM	COLP	COLS	COLT	COLZ	CPP	CPR	CRN	CRNPDB	CRNPRE	CSC
-	-	-	-	-	-	-	-	-	-	-	-	-				-
CSCQ	CSCX	CSR	CSVC	CTC	CUR	DXL	ECM	ECOE	ECP	ECPAUDIT	EDDBL	EDDBM	EDDBS	EDDH	EDDT	EMP
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
EQP	EQPSTAT	ESO	ESRR	ESRZ	FAC	FAX	FTD	FTDAUDIT	GAP	GBL	GENCAF	GIF	GSC	GSM	HLP	HMD
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
HRCAUDIT	IBF	IISX	IMGX	IML	INV	INVAUD	INVCAF	INVCMT	INVR	INVX	IPAS	ITC	IVRXFR	LGL	LHC	LNE
-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-

dataview

Endpoint Monitoring (continued)

Identify Queues with a backlog of messages pending processing
Identify Queues which are disabled

IMGX -	IML -	IPAS -	LNE -
PMT -	PND - OFF	PRD -	PREBIL -
SHPTMP -	SHPX 545	SLI 134	SLIPDB -
TSK -	TWG -	VALUES -	WEB -

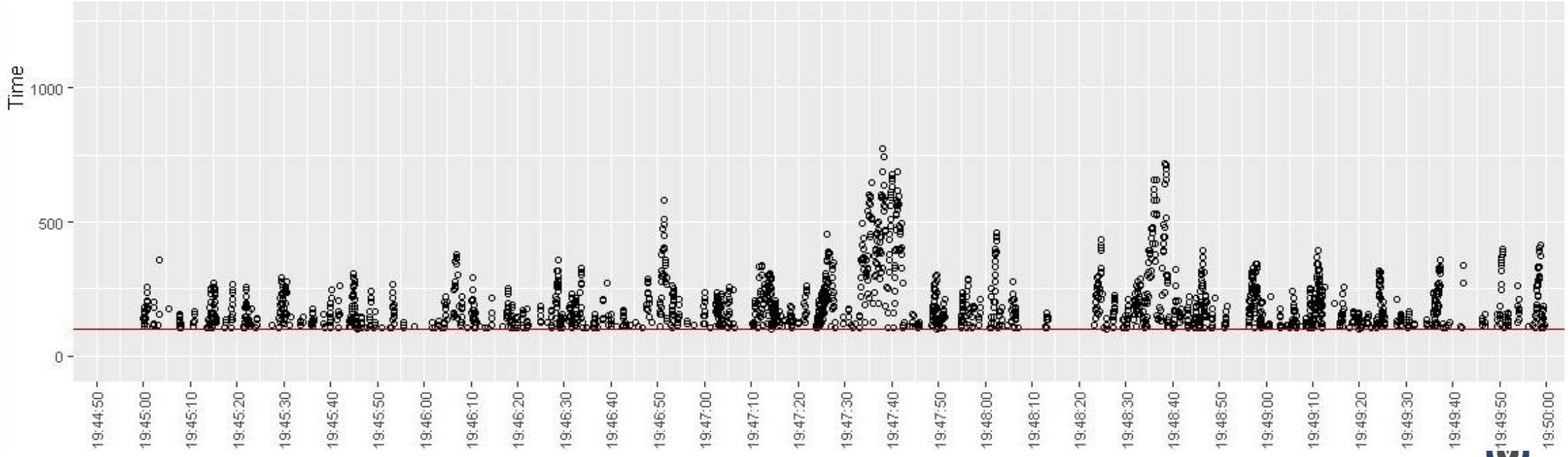
Max Reader Count per Queue

SLIPRE -	SPR -
WGP -	WRK -

File Group ID: 122
Max Reader = 20

Dashboards are great, but deep analysis is a must

- In a high velocity environment spikes in processing time can be washed out by only looking at aggregated values.
- This scatter plot shows that the median message processing time is 100 milliseconds (red line), however when looking at the metrics for individual executions we see spikes up to 750 milliseconds.

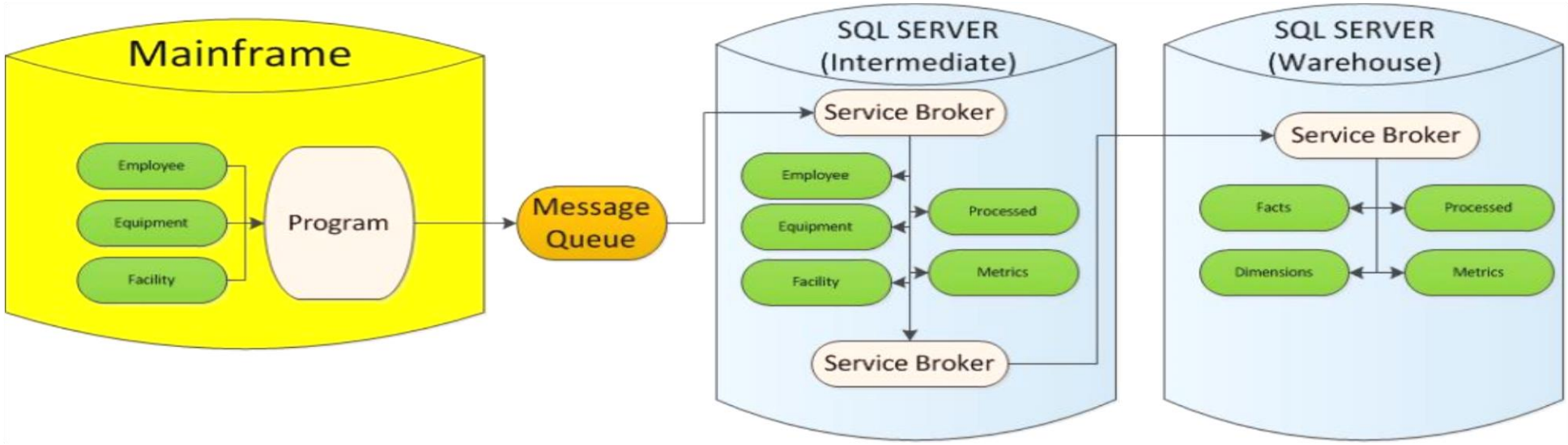


dataview

Extra Slides

Data Flow :: Mainframe

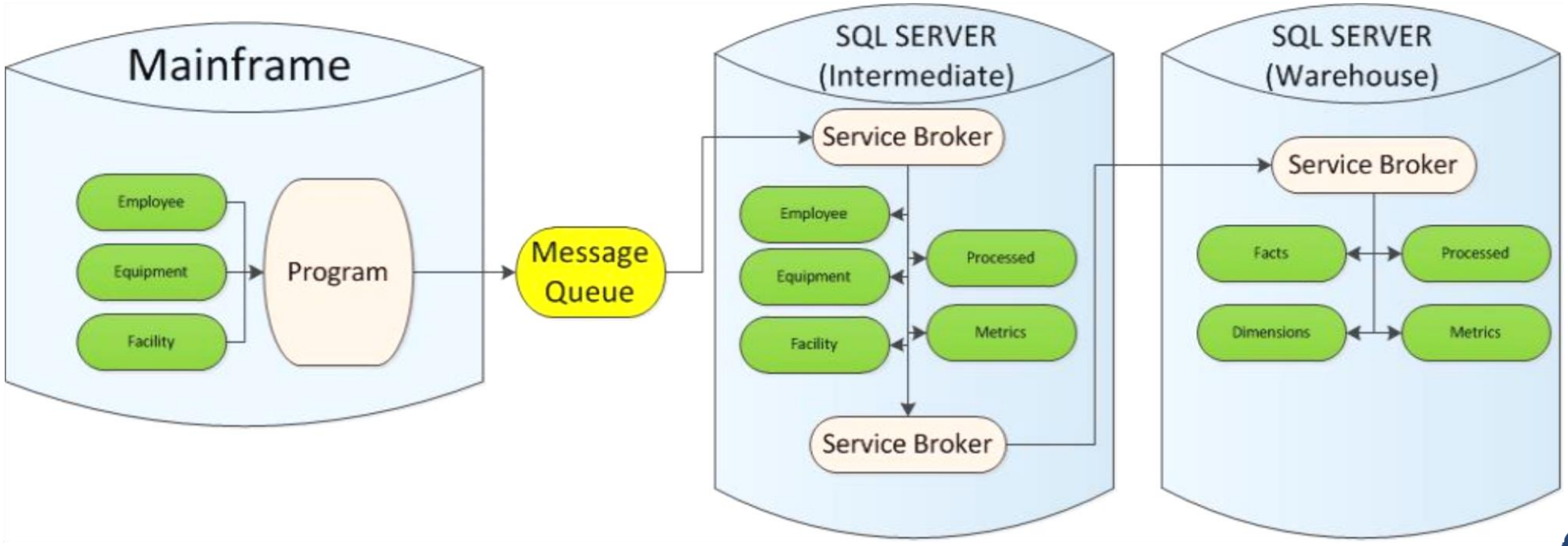
- Data modifications (Insert, Update and Delete) which occur within a Mainframe Program generate XML Messages after a commit action. Each XML Message contains the full version of the current record.



dataview

Data Flow :: External Message Queue

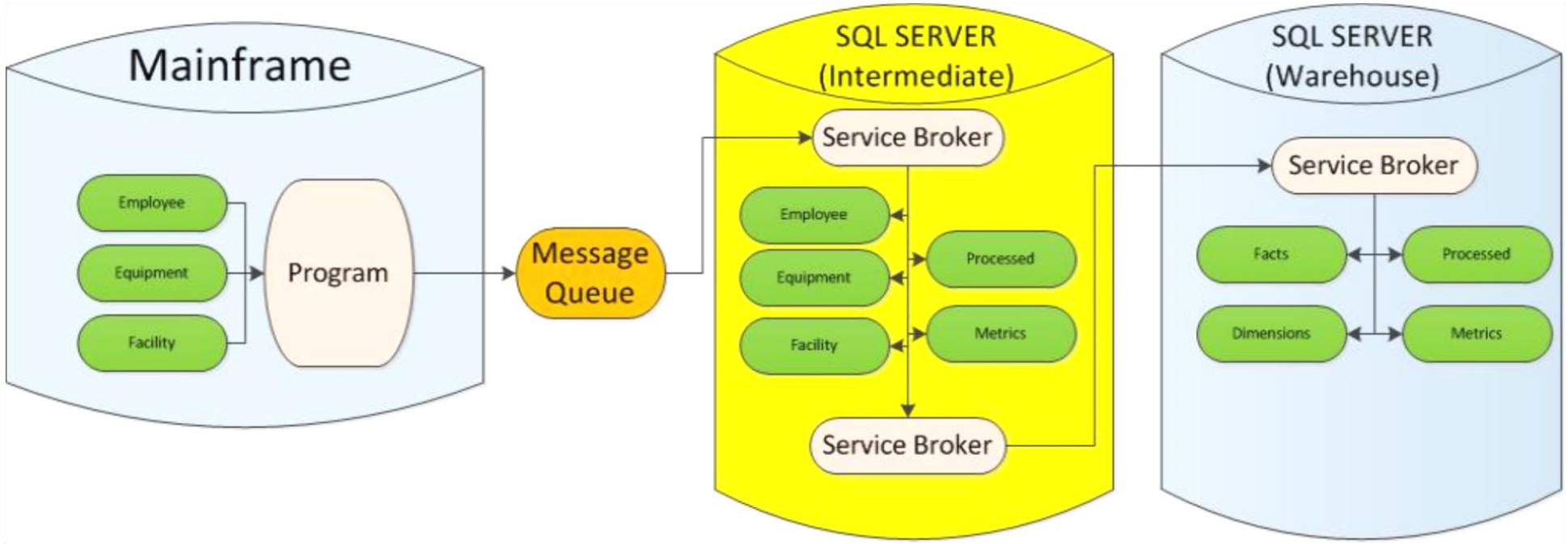
- The External Message Queue batches up individual messages (into groups of (n) messages) and then delivers the batched messages to Service Broker. This is step is done to avoid single record processing within SQL Server. It also provides resiliency in the process should SQL Server become unavailable.



dataview

Data Flow :: Intermediate Database

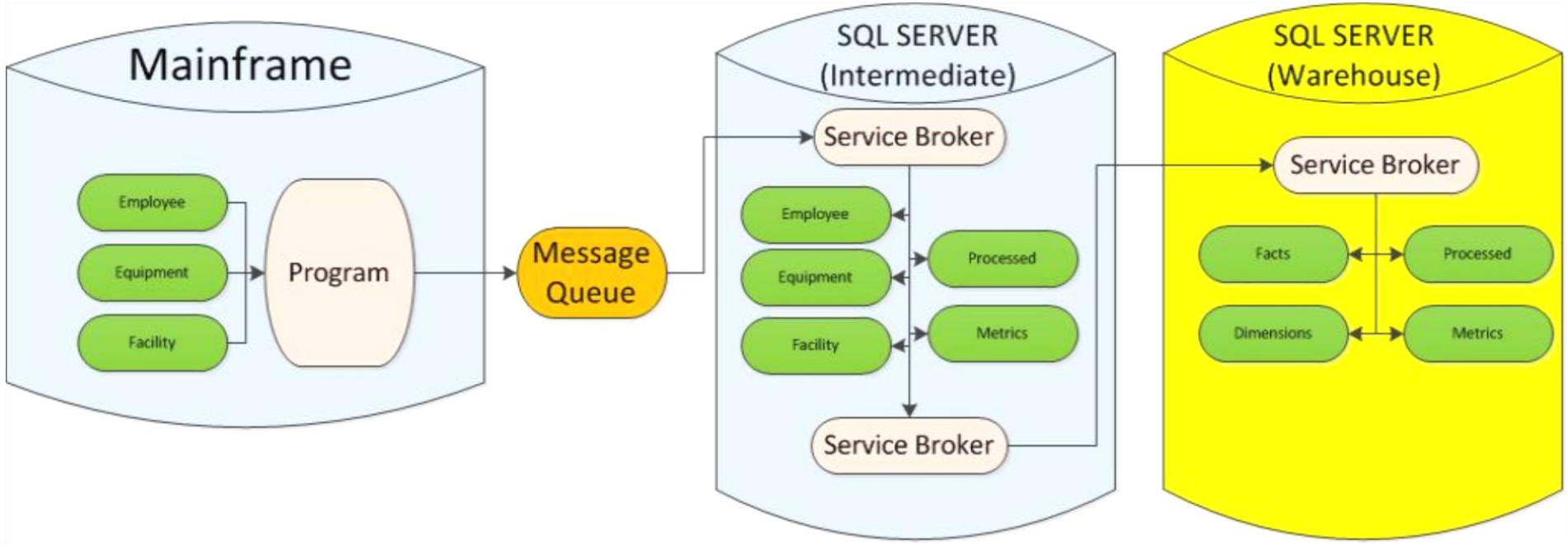
- External Message Queue writes the XML Messages to Service Broker
- The Service Broker queue activates a Stored Procedure which parses the XML, sorts the records and identifies the most recent record version in instances when we receive multiple updates within a single message batch
- The Stored Procedure then persists the data and logs the activity to the Processed and Metrics tables
- The original XML Message is stored in the Processed table for auditing purposes



dataview

Data Flow :: SQL Server Warehouse

- Cross Server Broker Queues deliver data from the Intermediate Database post processing.
- The Activation Stored Procedure in the Warehouse Transforms and Types the data which is then persisted in a typical Fact and Dimension model. The processed data is once again logged to a processed and metrics table.



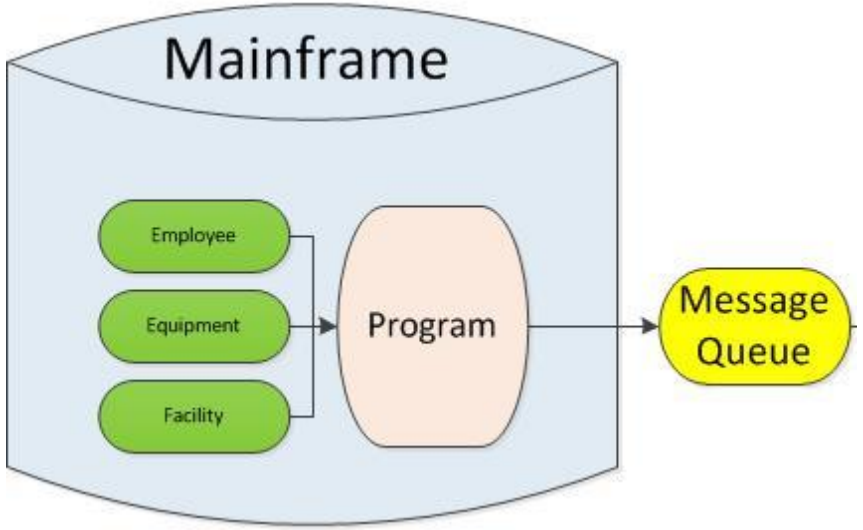
dataview

Batch Processing

The Mainframe sends individual messages to the External Message Queue.

The External Message Queue will attempt to batch (n) messages into a single message.

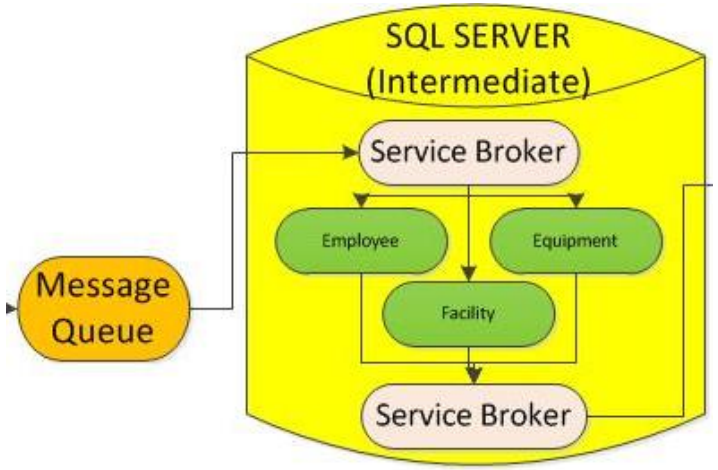
This batching reduces the number of connections to SQL Server and allows for further Batch Processing downstream.



Batch Processing (2)

The External Message Queue sends XML messages composed of (n) individual XML messages to Service Broker.

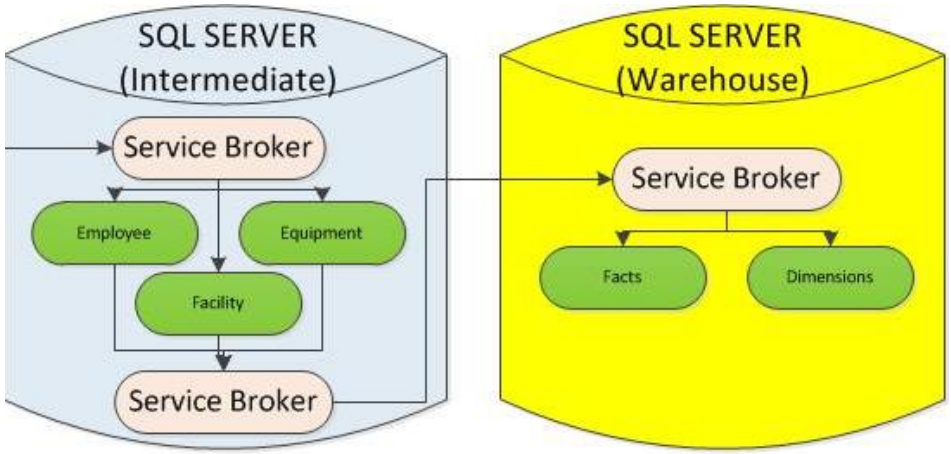
Messages are received from Service Broker in batches for processing.



Batch Processing (3)

The Cross Server Broker Queue sends the batch of messages processed in the Intermediate Database to the Target Queue in the Warehouse.

The messages on the Warehouse Queue are further pulled off in batches for processing.



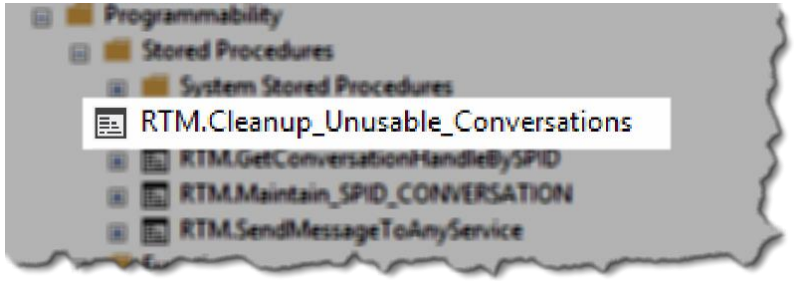
A better way to send… (5)


```
ALTER PROCEDURE RTM.GetConversationHandleBySPID
  TRY
    IF @TransactionCount = 0
      BEGIN TRANSACTION
    IF ( SELECT (Round ((Rand () * (@RandomExecutionRate - 1)), 0) + 1) ) = 1
      TRY
        EXECUTE RTM.Maintain_SPID_CONVERSATION
      CATCH
        IF Error_Number() = 1222
          EXECUTE RTM.AddEvent
        ELSE
          EXECUTE RTM.AddEvent
    SELECT FROM RTM.SPID_CONVERSATION, sys.conversation_endpoints
    IF @ch IS NULL
      BEGIN DIALOG CONVERSATION
      INSERT INTO RTM.SPID_CONVERSATION
      IF ( SELECT (Round ((Rand () * (10 - 1)), 0) + 1) ) = 1
        TRY
          EXECUTE RTM.Cleanup_Unusable_Conversations
        CATCH
          EXECUTE RTM.AddEvent
    IF @TransactionCount = 0
      COMMIT TRANSACTION
    RETURN
  CATCH
    EXECUTE RTM.AddEvent
    IF ( Xact_State() ) = -1
    IF ( Xact_State() ) = 1 AND @TransactionCount = 0
    IF ( Xact_State() ) = 1 AND @TransactionCount > 0
    INSERT INTO RTM.Process_M204_ErrorLog
```

```
ALTER PROCEDURE [RTM].[GetConversationHandleBySPID] (
    @InitiatorService VARCHAR(250)
   ,@TargetService VARCHAR(250)
   ,@Contract VARCHAR(250)
   ,@ch UNIQUEIDENTIFIER OUTPUT )
AS
```

dataview

A better way to send… (8)




```
ALTER PROCEDURE RTM.Cleanup_Unusable_Conversations
    IF @SessionID IS NULL
    TRY
        IF @TransactionCount = 0
            BEGIN TRANSACTION
        SELECT FROM @convolist
        WHILE @maxrow >= @count
            SELECT FROM @convolist
            TRY
                SELECT FROM @convolist @convo
            CATCH
                IF Error_Number() = 8426
                    PRINT
        IF @maxrow >= 1
        IF @TransactionCount = 0
            COMMIT TRANSACTION
        EXECUTE RTM.AddEvent
    CATCH
        EXECUTE RTM.AddEvent
        IF ( Xact_State() ) = -1
        IF ( Xact_State() ) = 1 AND @TransactionCount = 0
        IF ( Xact_State() ) = 1 AND @TransactionCount > 0
```

```
ALTER PROCEDURE [RTM].[Cleanup_Unusable_Conversations] (
    @SessionID UNIQUEIDENTIFIER = NULL )
AS
```

dataview

A better way to send… (11)


```
SET DEADLOCK_PRIORITY LOW
SET LOCK_TIMEOUT 1000
```

```
SELECT
        [sc].[ID]
       ,[sc].[SPID]
       ,[sc].[INITIATOR_SERVICE]
       ,[sc].[TARGET_SERVICE]
       ,[sc].[CONTRACT]
       ,[sc].[CONVERSATION_HANDLE]
    FROM
        [RTM].[SPID_CONVERSATION] [sc] WITH ( READPAST )
    LEFT JOIN [sys].[conversation_endpoints] [ce] WITH ( READPAST ) ON [sc].[CONVERSATION_HANDLE] = [ce].[conversation_handle] AND [ce].[state] IN ( 'CO', 'SO' )
    WHERE
        [ce].[conversation_handle] IS NULL;
```

Finding conversations that are not in a valid state

Belts and Braces: Lock_Timeout = 1000 and READPAST hint keep us from waiting for other consumers

```
DELETE
        [sc]
    FROM
        [RTM].[SPID_CONVERSATION] [sc]
    JOIN @convolist [c] ON [sc].[ID] = [c].[ID];
```

dataview

What happens when things go wrong?

Activation SP :: Retry, “expected errors”

- Used for retry for selected error conditions within the activation procedure where recovery is possible
- Disk Based Tables Only:
  - `IF ERROR_NUMBER() IN ( 1205, 2627 ) --1205 Deadlock, 2627 Primary Key Violation`
- Additional Criteria for Memory Tables:
  - `ERROR_NUMBER() IN ( 41302, 41305, 41325, 41301)`
- Data typing into the data warehouse is handled proactively

dataview

Activation SP :: Retry, “expected errors”

- Uses @MaxRetry, @WaitForDelay from external configuration
- @WaitForDelay provides time for deadlocks and write conflicts from other broker threads to resolve
  - Typically 100 Milliseconds
- Excellent MSDN reference: https://msdn.microsoft.com/en-us/library/dn169141.aspx

```
SET @retry = @MaxRetry;
WHILE ( @retry > 0 )
    BEGIN
        BEGIN TRY
            BEGIN TRANSACTION;

            --Attempt Transactions into table

            COMMIT TRANSACTION;

            SET @retry = 0;
        END TRY
        BEGIN CATCH
            SET @retry -= 1;

            IF (   @retry > 0
                   AND ERROR_NUMBER() IN ( 41302, 41305 ,  41325, 41301))
                BEGIN
                    -- these error conditions are transaction dooming - rollback the transaction
                    IF XACT_STATE() = -1
                        ROLLBACK TRANSACTION;

                    -- use a delay if there is a high rate of write conflicts (41302)
                    WAITFOR DELAY @WaitForDelay;
                END;
            ELSE
                BEGIN
                    -- insert custom error handling for other error conditions here
                     ;
                    THROW;
                END;
        END CATCH;
    END;
```

dataview

Activation SP :: “unexpected” Error Patterns

- Typical Error scenario is one where a row within a batch fails.
- Multiple patterns exists, depending on the use case:
  - Write entire batch to Error Table and reprocess with batch job (15 minute interval)
  - Cursor through batch and recursively call Stored Procedure
  - Cursor through batch, resending to same queue with elevated broker priority
  - Send to exception queue using “binary search” send. Resuse same activation procedure but only process single broker message in each receive from queue.

Activation SP :: SQL Parsing of XML

Initially the messages were parsed using TSQL —this was found to be resource-intensive (TempDB)

Resource consumption becomes a real issue when dealing with records with a large number of Multiply Occurring Fields (MOFs)

```
        )
        SELECT [RECORD_KEY] = [b].[value]('payload[1]/record[1]/RECORD.KEY[1]', 'numeric (30,0)'),
               [CAR_REASON_CD] = [b].[value]('payload[1]/record[1]/CAR.REASON.CD[1]', 'varchar(255)'),
               [CAR_WRKGRP] = [b].[value]('payload[1]/record[1]/CAR.WRKGRP[1]', 'varchar(255)'),
               [CARE_SHP_TERMS] = [b].[value]('payload[1]/record[1]/CARE.SHP.TERMS[1]', 'varchar(255)'),
               [CARS_AUDIT_ACTION] = [b].[value]('payload[1]/record[1]/CARS.AUDIT.ACTION[1]', 'varchar(255)'),
               [CARS_AUDIT_DATE] = [b].[value]('payload[1]/record[1]/CARS.AUDIT.DATE[1]', 'varchar(255)'),
               [CARS_AUDIT_TIMESTAMP] = [b].[value]('payload[1]/record[1]/CARS.AUDIT.TIMESTAMP[1]', 'varchar(255)'),
               [CARS_AUDITOR] = [b].[value]('payload[1]/record[1]/CARS.AUDITOR[1]', 'varchar(255)'),
               [CARS_AUTORATE_IND] = [b].[value]('payload[1]/record[1]/CARS.AUTORATE.IND[1]', 'varchar(255)'),
               [CARS_BATCH_QUEUE] = [b].[value]('payload[1]/record[1]/CARS.BATCH.QUEUE[1]', 'varchar(255)'),
               [CARS_BILLED_TIMESTAMP] = [b].[value](
                                                     'payload[1]/record[1]/CARS.BILLED.TIMESTAMP[1]',
                                                     'varchar(255)'
                                                   ),
               [CARS_BUS_CMPNY_TYPE] = [b].[value]('payload[1]/record[1]/CARS.BUS.CMPNY.TYPE[1]', 'varchar(255)'),
               CASE
                   WHEN [b].[value]('payload[1]/record[1]/CARS.CONSG.ATTN.LINE[1]/@encoding[1]', 'varchar(50)') IS NULL THEN
                       [b].[value]('payload[1]/record[1]/CARS.CONSG.ATTN.LINE[1]', 'VARCHAR(255)')
                   ELSE
                       LEFT([dbo].[TranslateB64EbcdicToUTF8]([b].[value](
                                                                         'payload[1]/record[1]/CARS.CONSG.ATTN.LINE[1]',
                                                                         'VARCHAR(MAX)'
                                                                        )
                                                                       ), 255)
               END AS [CARS_CONSG_ATTN_LINE],
               [CARS_CONSG_BUS_NM] = [b].[value]('payload[1]/record[1]/CARS.CONSG.BUS.NM[1]', 'varchar(255)'),
               [CARS_CONSG_PLC_ADDR_STREET] = [b].[value](
                                                          'payload[1]/record[1]/CARS.CONSG.PLC.ADDR.STREET[1]',
                                                          'varchar(255)'
                                                         ),
               [CARS_CONSG_RGN_CITY] = [b].[value]('payload[1]/record[1]/CARS.CONSG.RGN.CITY[1]', 'varchar(255)'),
               [CARS_CONSG_RGN_CTRY_ABBR] = [b].[value](
                                                        'payload[1]/record[1]/CARS.CONSG.RGN.CTRY.ABBR[1]',
                                                        'varchar(255)'
                                                       ),
               [CARS_CONSG_RGN_ST_ABBR] = [b].[value](
                                                      'payload[1]/record[1]/CARS.CONSG.RGN.ST.ABBR[1]',
                                                      'varchar(255)'
```


dataview

Activation SP :: Parsing the XML Message

The messages now are parsed using C# and .NET XmlReader, invoked from SQL

```
)
SELECT [RECORD_KEY] = CAST([b].[RECORD_KEY] AS NUMERIC(30, 0)),
       [DTS_DTTMSP] = [b].[DTS_DTTMSP],
       [CAR_REASON_CD] = CAST([b].[CAR_REASON_CD] AS VARCHAR(255)),
       [CAR_REASON_REF_NBR] = CAST([b].[CAR_REASON_REF_NBR] AS VARCHAR(255)),
       [CAR_WRKGRP] = CAST([b].[CAR_WRKGRP] AS VARCHAR(255)),
       [CARE_SHP_TERMS] = CAST([b].[CARE_SHP_TERMS] AS VARCHAR(255)),
       [CARS_AUDIT_ACTION] = CAST([b].[CARS_AUDIT_ACTION] AS VARCHAR(255)),
       [CARS_AUDIT_COMMENTS] = CAST([b].[CARS_AUDIT_COMMENTS] AS VARCHAR(255)),
       [CARS_AUDIT_DATE] = CAST([b].[CARS_AUDIT_DATE] AS VARCHAR(255)),
       [CARS_AUDIT_TIMESTAMP] = CAST([b].[CARS_AUDIT_TIMESTAMP] AS VARCHAR(255)),
       [CARS_AUDITOR] = CAST([b].[CARS_AUDITOR] AS VARCHAR(255)),
       [CARS_AUTORATE_IND] = CAST([b].[CARS_AUTORATE_IND] AS VARCHAR(255)),
       [CARS_BATCH_QUEUE] = CAST([b].[CARS_BATCH_QUEUE] AS VARCHAR(255)),
       [CARS_BILLED_TIMESTAMP] = CAST([b].[CARS_BILLED_TIMESTAMP] AS VARCHAR(255)),
       [CARS_BUS_CMPNY_TYPE] = CAST([b].[CARS_BUS_CMPNY_TYPE] AS VARCHAR(255)),
       [CARS_CONSG_ATTN_LINE] = CAST([b].[CARS_CONSG_ATTN_LINE] AS VARCHAR(255)),
       [CARS_CONSG_BUS_NM] = CAST([b].[CARS_CONSG_BUS_NM] AS VARCHAR(255)),
       [CARS_CONSG_PLC_ADDR_STREET] = CAST([b].[CARS_CONSG_PLC_ADDR_STREET] AS VARCHAR(255)),
       [CARS_CONSG_RGN_CITY] = CAST([b].[CARS_CONSG_RGN_CITY] AS VARCHAR(255)),
       [CARS_CONSG_RGN_CTRY_ABBR] = CAST([b].[CARS_CONSG_RGN_CTRY_ABBR] AS VARCHAR(255)),
       [CARS_CONSG_RGN_ST_ABBR] = CAST([b].[CARS_CONSG_RGN_ST_ABBR] AS VARCHAR(255)),
       [CARS_CONSG_RGN_ZIP] = CAST([b].[CARS_CONSG_RGN_ZIP] AS VARCHAR(255)),
       [CARS_CONSG_SLS_ORZ] = CAST([b].[CARS_CONSG_SLS_ORZ] AS VARCHAR(255)),
       [CARS_CONSG_STORE_ID] = CAST([b].[CARS_CONSG_STORE_ID] AS VARCHAR(255)),
       [CARS_CORRECTION_IND] = CAST([b].[CARS_CORRECTION_IND] AS VARCHAR(255)),
       [CARS_CRRNCY_CD] = CAST([b].[CARS_CRRNCY_CD] AS VARCHAR(255)),
       [CARS_DELIVERED_TIMESTAMP] = CAST([b].[CARS_DELIVERED_TIMESTAMP] AS VARCHAR(255)),
```

```
       [A].[message_enqueue_time],
       [A].[payload],
       [ProcessFlag] = 0,
       [ProgramId] = CAST([b].[programId] AS VARCHAR(255)),
       [encoded] = [b].[encoded]
FROM @Messages_Temp [A]
    CROSS APPLY [RTM].[ParseBaseCARSMessage](CAST([A].[payload] AS NVARCHAR(MAX))) AS [b];
```


dataview

If you don’t write it down it didn’t happen…

- Payload logging
- Events
- Errors
- Metrics

dataview

Troubleshooting

- Transmission Queue is very important, and you must have DBO permissions to see
- Conversation management and cleanup
- Routes
- Service Permissions
- Alerting

dataview

Shard data to avoid deadlocking

- Create a Partition Scheme based on numeric Record Key

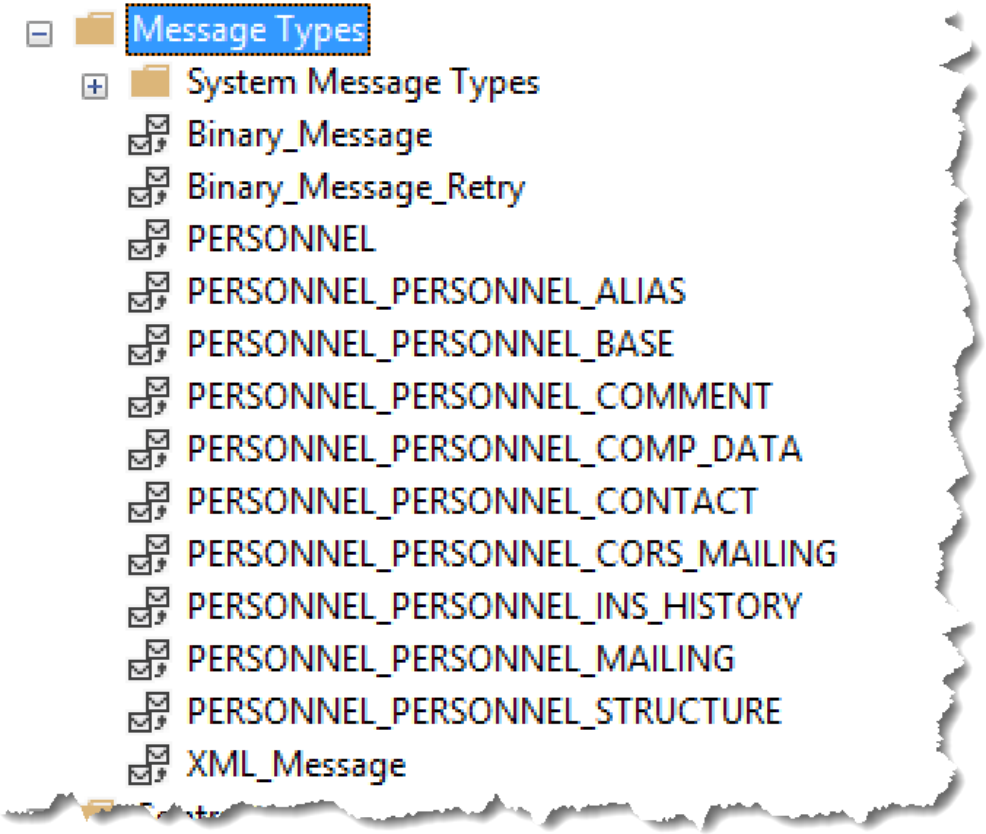
Some of the major challenges faced

- Conversation Management
  - Conversation Locking / deadlocking
  - Message loss thru conversations ending
  - Developers correctly using “send”
  - Transaction Management while using convos and “send”
- Error Handling
- Ensuring Delivery / Receipt Cross server
- Balancing Multithreaded workload into tables
- Maximum number of threads running across queues
- Maximum number of total queue readers on database
- hardware was at a minimum due to client budget constraints
- XML parsing using TSQL vs C#
- SQL code databases separate from payload and data

dataview

Multiple Message Type Use

- Service Broker Metrics may be setup
- For determining where to land messages in DW
  - (Equipment, Facility, Personnel)
- For indicating retry and/or priority of messages



dataview

Intermediate Database Table Design

- Key features of the Intermediate Database table design are as follows:
  - Records for each File Group are split into no more than two tables (Base and Reoccurring)
  - All fields are varchar(255) and nullable with the exception of RECORD_KEY and numeric fields used as part of an index
    - The decision to use varchar was made to ensure we do not lose any records due data conversion errors
    - Data in the XML message is all text so there are no conversion issues when defining the target field as varchar
    - The (255) length was chosen because this is generally the max field length on the Mainframe. This ensures we do not truncate data.
    - Fields can by typed and sized when processed into the Warehouse
  - The decision to not do any data transformation in this step allows for easier data validation between the Mainframe and the Intermediate Database . Any data transformations would need to be reversed before validation could occur.
- This design allows us to generate our table structures and code using a code generator

Activation SP :: Data Typing into DW

```
entire batch base records
INTO [#b]
[S].*
(   SELECT [RECORD_KEY] = [B].[RECORD_KEY]
          ,[DTS_DTTMSP] = [B].[DTS_DTTMSP]
          ,[Create_TS] = [B].[Create_TS]
          ,[Modify_TS] = [B].[Modify_TS]
          ,[Modify_ID] = [B].[Modify_ID]
          ,[BUS_TAG] = [B].[BUS_TAG]
          ,[CREATE_DATE] = CAST([THB_Stage].[MFD1_FormatDate]([B].[CREATE_DATE]) AS DAT
          ,[DATE_R] = CAST([THB_Stage].[MFD1_FormatDate]([B].[DATE]) AS DATE)
          ,[FUNCTION_R] = CAST([B].[FUNCTION] AS CHAR(1))
          ,[PROCEDURE_R] = [B].[PROCEDURE]
          ,[TIME_R] = CAST([THB_Stage].[MFD2_FormatTime]([B].[TIME]) AS TIME(7))
          ,[USER_R] = [B].[USER]
          ,[Source_Deleted_IND] = [B].[SOURCEDELETED]
    FROM   @base AS [B]
    LEFT JOIN @reoccur AS [R]
        ON [B].[RECORD_KEY]     = [R].[IISX_RECORD_KEY]
           AND [B].[DTS_DTTMSP] = [R].[DTS_DTTMSP]
           AND [R].[OCCNO]      = 1
) AS [S]
[S].[Source_Deleted_IND] = 0;
```

CAST the data to the correct type for the batch

- CAST the entire message batch to the correct data types
- If all's well persist the records

dataview

Activation SP :: Intermediate DB

- Key design features of the Intermediate DB Activation SP
  - External Configuration
  - While Loop
  - Logging to Processed Table
  - Error Handling and Logging
  - Metrics Logging

```
SELECT FROM @v
WHILE @LoopCounter < @MaxLoopCounter
  BEGIN TRANSACTION
  TRY
    RECEIVE FROM EQP_TargetQueue
    IF @@ROWCOUNT = 0
    COMMIT TRANSACTION
  CATCH
  IF ( SELECT COUNT(1) FROM @tableMessages ) >= 1
    TRY
      IF ( SELECT COUNT(1) FROM @MOF ) >= 1
      IF EXISTS (SELECT COUNT(1) FROM @Messages WHERE [ProcessFlag] = 1)
        UPDATE M
        WHILE @InsertRetryCount < @MaxRetry AND @InsertSuccess = 0
          TRY
            BEGIN TRANSACTION
            INSERT INTO Legacy.Mainframe_EQP
            COMMIT TRANSACTION
          CATCH
        WHILE @UpdateRetryCount < @MaxRetry AND @UpdateSuccess = 0
          TRY
            BEGIN TRANSACTION
            UPDATE T
            COMMIT TRANS
```

dataview

Activation SP :: Data Typing into DW

```
  SYSDATETIME()
SELECT [RECORD_KEY] = [B].[RECORD_KEY]
      ,[DTS_DTTMSP] = [B].[DTS_DTTMSP]
      ,[Create_TS] = [B].[Create_TS]
      ,[Modify_TS] = [B].[Modify_TS]
      ,[Error] = CASE WHEN [dbo].[IsColumnPopulated](TRY_CAST([B].[Modify_ID] AS VARCHAR(255))) =
                          [dbo].[IsColumnPopulated]([B].[Modify_ID]) THEN
                      NULL
                  ELSE
                      'failure casting column [Modify_ID],  value: '
                      + CAST(ISNULL([B].[Modify_ID], 'NULL') AS VARCHAR(255))
                END
      ,[Error] = CASE WHEN [dbo].[IsColumnPopulated](TRY_CAST([B].[BUS_TAG] AS VARCHAR(255))) =
                          [dbo].[IsColumnPopulated]([B].[BUS_TAG]) THEN
                      NULL
                  ELSE
                      'failure casting column [BUS_TAG],  value: '
                      + CAST(ISNULL([B].[BUS_TAG], 'NULL') AS VARCHAR(255))
                END
      ,[Error] = CASE WHEN [dbo].[IsColumnPopulated](TRY_CAST([THB_Stage].[MFD1_FormatDate]([B].[CREATE_DATE])
                          [dbo].[IsColumnPopulated]([B].[CREATE_DATE]) THEN
                      NULL
                  ELSE
                      'failure casting column [CREATE_DATE],  value: '
                      + CAST(ISNULL([B].[CREATE_DATE], 'NULL') AS VARCHAR(255))
                      + ' as DATE'
                END
      ,[Error] = CASE WHEN [dbo].[IsColumnPopulated](TRY_CAST([THB_Stage].[MFD1_FormatDate]([B].[DATE]) AS DAT
                          [dbo].[IsColumnPopulated]([B].[DATE]) THEN
                      NULL
                  ELSE
                      'failure casting column [DATE_R],  value: '
                      + CAST(ISNULL([B].[DATE], 'NULL') AS VARCHAR(255))
```

CAST messages one by one to the correct data type

- Iterate through the messages
- CAST the data to the correct data types
- Persist those records that were correctly typed
- Write those errors with type mismatch to error table

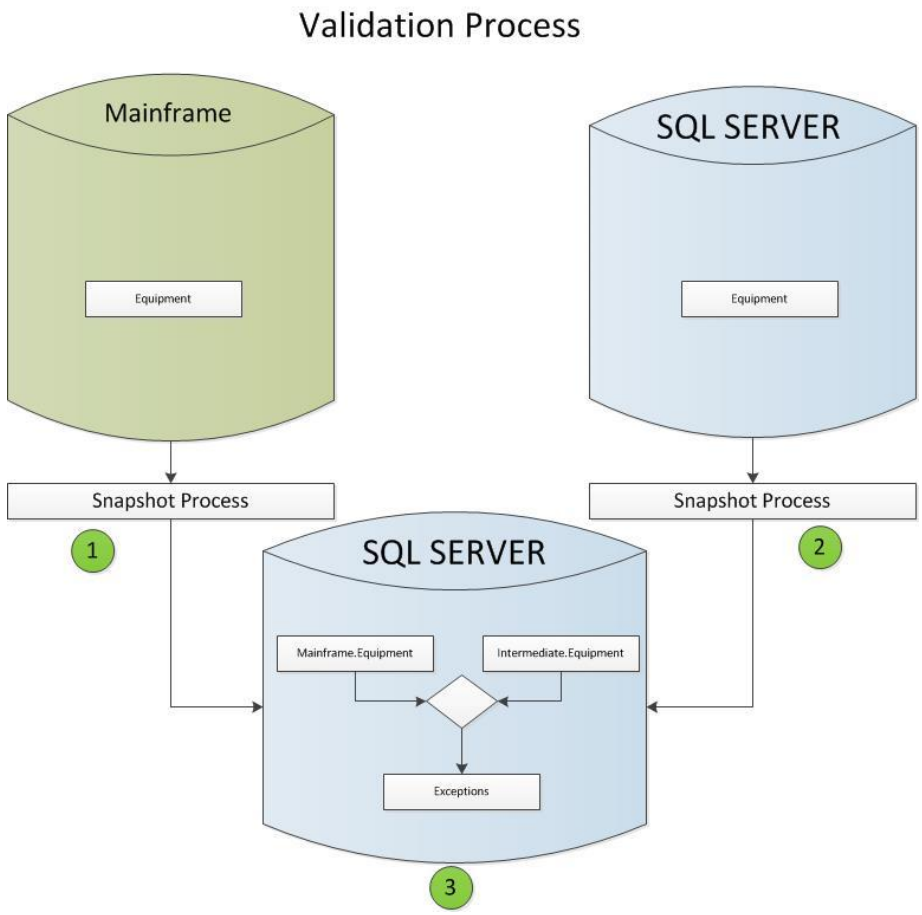
dataview

Data Validation

- 1) A Snapshot is taken of Mainframe Data Files during the morning “bounce”. The data from the static snapshot is loaded into SQL Server tables which match the structure of the Intermediate Database tables (For large files you may need to limit to recently updated records)

- 2) A Snapshot is taken of the Intermediate Database which corresponds to the Mainframe Snapshot. This data is also loaded into a the same database as the Mainframe snapshot data

- 3) The Mainframe data is compared against Intermediate Database data. Any inconsistencies are logged into an exception table and used in a re-synch process



Data Validation Report

The validation report shows the data accuracy between Mainframe and the Intermediate Database.

The bulk of records that do not match are due to transactions which were backed out of the Mainframe after the update was sent to the Intermediate Database

Table	Validation Date	Record Count	Mismatch Count	Data Accuracy %
	4/26/2016	1,215,687	17	99.999%
	4/26/2016	13,519,836	20	100.000%
	4/26/2016	33,132	0	100.000%
	4/26/2016	51,788	2	99.996%
	4/26/2016	214,495	2	99.999%
	4/26/2016	200,004	0	100.000%
	4/26/2016	2,653,561	0	100.000%
	4/26/2016	120,936	64	99.996%
	4/26/2016	46,091	0	100.000%
	4/26/2016	24,860	0	100.000%
	4/26/2016	157,871	0	100.000%
	4/26/2016	13,115	0	100.000%
	4/26/2016	1,880,954	2	100.000%
	4/26/2016	2,564,692	32	99.999%
	4/26/2016	1,504,728	47	99.997%
	4/26/2016	3,494,014	42	99.999%
	4/26/2016	99,164	0	100.000%
	4/26/2016	476,693	0	100.000%
	4/26/2016	15,863	0	100.000%
	4/26/2016	298	0	100.000%
	4/26/2016	900,655	2	100.000%
	4/26/2016	2,886	0	100.000%
	4/26/2016	4,250	0	100.000%