



| WebSphere Data Interchange

## 2005 B2B Customer Conference

*Pioneering New Horizons – Solutions that Evolve*

### Managing Your WDI Environment

| David Hixon  
Product Architect, WDI and WPG

October 2005

© 2005 IBM Corporation



## Agenda

- **Environments**
- **Log Files**
- **\* Transaction Store**
- **\* Alerts**
- **Replay/Resend**
- **\* Overdue Acknowledgments**
- **Other topics: monitoring, backup/restore**



## Environments

- **Batch**
  - JCL
  - ediservr
- **Real Time**
  - CICS
  - MQ Triggering
- **API**
  - C++ and Java
  - Utility



| WebSphere Data Interchange

## Using and Managing Log Files

| Separating logs by application

October 2005

© 2005 IBM Corporation



## Using and Managing Log Files

- **Issues with log files**
- **Tools and techniques for dealing with log files**
  - Logical logs
  - Pruning logs
  - Restoring logs
  - Reducing the number of logged messages
  - Securing the logs
  - Correlating logs
  - Operating without the Event Log
- **Best practices discussion**



## Issues with log files

- **They fill up or get too big and need to be pruned (archived)**
- **How can you view events that have been archived?**
- **How can you archive such that events can be found and restored?**
- **How can you correlate log entries with the PRTFILE**
- **They could contain confidential information**

## Logical Logs

- **Why**
  - Security
  - Easier problem determination?
  - Allow archiving on different schedules
- **How logical logs are implemented**
  - Logical name in a single DB2 table
- **How to specify different log files**
  - APPLID parameter on EDIFFS
  - APPLID property in the properties file

## Creating Logical Logs

Local DB2 - Activity Log - EDIFFS

General Comments

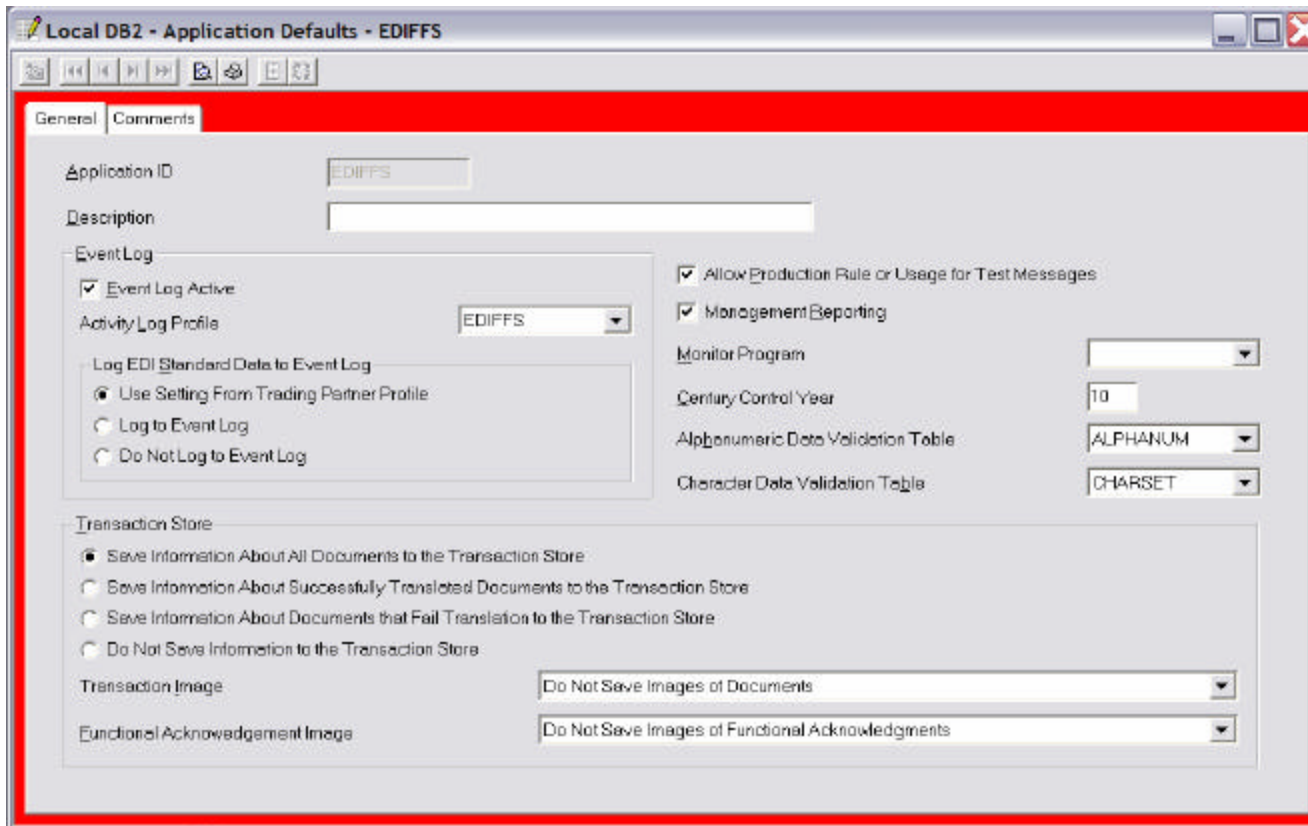
Application ID: EDIFFS

Description: [Empty text box]

Activate the Log:



## Configuring Applications to Use Logical Logs



Local DB2 - Application Defaults - EDIFFS

General Comments

Application ID: EDIFFS

Description:

Event Log

Event Log Active

Activity Log Profile: EDIFFS

Log EDI Standard Data to Event Log

Use Setting From Trading Partner Profile

Log to Event Log

Do Not Log to Event Log

Allow Production Rule or Usage for Test Messages

Management Reporting

Monitor Program:

Century Control Year: 10

Alphanumeric Data Validation Table: ALPHANUM

Character Data Validation Table: CHARSET

Transaction Store

Save Information About All Documents to the Transaction Store

Save Information About Successfully Translated Documents to the Transaction Store

Save Information About Documents that Fail Translation to the Transaction Store

Do Not Save Information to the Transaction Store

Transaction Image: Do Not Save Images of Documents

Functional Acknowledgement Image: Do Not Save Images of Functional Acknowledgments

## Pruning the Log

- **UNLOAD LOG ENTRIES command**
- This command reads all the entries in the event log associated with the Application ID. The entries selected for removal are copied to the archive file, and all other entries associated with the Application ID are copied to the hold file. Event log entries that are associated with active or held transactions in the Transaction Store are not eligible for archive and are not selected for removal. The entries selected for removal are copied to the archive file and immediately deleted. To ensure concurrency, set the number of deletes (Numdels) performed before a COMMIT is issued to a relatively low value. This command may be run as the first step of a multi-step process. (See the table on 5.)
- **APPLID**(*application ID*)
- **ARCHIVEFILE**(*event log archive file name*)
- **ARCHIVETYPE**(*event log archive file type*)
- **HOLDFILE**(*event log hold file name*)
- **HOLDTYPE**(*event log hold file type*)
- **IFCC**(*override condition codes*)
- **LOGAEID**(*starting event log associated entry ID*) **TO**(*ending event log associated entry ID*)
- **LOGDATE**(*starting event log date*) **TO**(*ending event log date*)
- **LOGFORM**(*starting event log format ID*) **TO**(*ending event log format ID*)
- **LOGTIME**(*starting event log time*) **TO**(*ending event log time*)
- **LOGUSER**(*starting event log user ID*) **TO**(*ending event log user ID*)
- **NUMDELS**(*number of database deletes before commit*)
- **SETCC**(*condition codes*)
- **Example:** Unload log entries from the application log file for application EDIFFS dated December 14, 2001.
- **PERFORM UNLOAD LOG ENTRIES WHERE APPLID(EDIFFS) LOGDATE(12/14/01) ARCHIVEFILE(ARCHTRAN) HOLDFILE (HOLDTRAN)**

## Restoring Log Entries

- **LOAD LOG ENTRIES** command
- This command copies the selected HOLDFILE records back into the event log table. In a DB2 environment, you can restore deleted records to the event log by specifying the ARCHIVEFILE value from the UNLOAD LOG ENTRIES command as the value for the HOLDFILE keyword on this command.
- **APPLID**(*application ID*)
- **HOLDFILE**(*event log hold file name*)
- **HOLDTYPE**(*event log hold file type*)
- **IFCC**(*override condition codes*)
- **LOGAEID**(*starting event log associated entry ID*)
- **TO**(*ending event log associated entry ID*)
- **LOGDATE**(*starting event log date*) **TO**(*ending event log date*)
- **LOGFORM**(*starting event log format ID*) **TO**(*ending event log format ID*)
- **LOGTIME**(*starting event log time*) **TO**(*ending event log time*)
- **LOGUSER**(*starting event log user ID*) **TO**(*ending event log user ID*)
- **NEWAPPLID**(*new application ID*)
- **SETCC**(*condition codes*)
- **Example:** Copy the held records for **EDIFFS** with a log date of **12/14/01** back into the event log table.
- **PERFORM LOAD LOG ENTRIES WHERE APPLID(EDIFFS) LOGDATE(12/14/01)**

## Reducing the number of logged messages

- IGNOREWARN(Y) - Used for data transformation maps, it filters all warning messages. PERFORM TRANSFORM WHERE . . . IGNOREWARN(Y)
- FILTERMSGS(msg\_id, msg\_id) - For data transformation processing. Filters messages in the list, if severity is less than 8. Maximum length is 80. Up to 11 individual messages may be filtered. PERFORM TRANSFORM WHERE . . . FILTERMSGS(TR0016, TR0004)
- DIERRFILTER(msg\_id, . . . ) keyword - Specifies the initial set of errors to filter for this translation session. PERFORM DEENVELOPE AND TRANSLATE WHERE . . . DIERRFILTER(TR0016, TR0004)
- DIERRFILTER special variable – S/R mapping only. When SET or SAVED, the translator parses the value of the variable as an indicator of the errors to ignore. The DIERRFILTER variable value should consist of a list of the error codes to be ignored.
- Some objects can be added to eliminate messages, like NETCOMMANDS FSUPPORT, or a profile may be missing (but harmless)



## Securing the Logs

- **Use the DB2 view security technique to create views that can only see certain logical logs**
- **CREATE VIEW EDIBUYER.EDIELOG AS SELECT \* FROM EDIEC32E.EDIELOG WHERE APPLID = 'PURCHSNG'**

## Correlating Logs

- **On Windows the log can be correlated to the transaction store by matching on event log date and time with the transaction store created date and time**
- **On AIX and z/OS the log can be correlated to the transaction store by matching the transaction handle to the event ID**

## Operating without Logs

- **Pros**

- Can save time and improve throughput in environments that are “pushing the envelope” if turned off

- **Cons**

- Only way to detect certain database errors
- Only way to debug remotely using just the client (combination of event log and trx store)
- A more controlled and reliable repository for error msgs than PRTFILES



## Best Practices Discussion

- **Separate logs by application**
- **Filter warning messages**
- **Limit access using DB2 view technique**
- **Run periodic jobs to unload the logs**
- **Organize unload files by how they are searched**
- **Synchronize unloads with the transaction store**
- **If performance is not good enough, consider running without the event log**





WebSphere Data Interchange

## Managing the Transaction Store

Keeping the amount of data in the transaction store under control

October 2005

© 2005 IBM Corporation

## Managing the Transaction Store

- **Issues with the Transaction Store**
- **Tools and techniques for dealing with the Transaction Store**
  - Logical Transaction Stores
  - Pruning the Transaction Store
  - Restoring to the Transaction Store
  - Reducing the volume of data put to the Transaction Store
  - Securing the Transaction Store
  - Correlating events with the Transaction Store
  - Operating without the Transaction Store
- **Best practices discussion**



## Issues with the Transaction Store

- **It fills up or gets too big and need to be pruned (archived)**
- **How can you view messages that have been archived?**
- **How can you archive such that messages can be found and restored?**
- **How can you correlate messages with log entries and with the PRTFILE**
- **It could contain confidential information**

## Logical Transaction Stores

- **Why**
  - Security
  - Deadlocks and Timeouts
  - Allow archiving on different schedules
- **How logical transaction stores are implemented**
  - Replicate the transaction store tables
- **How to specify different transaction stores**
  - Use DB2 aliases similar to how security is implemented
  - Control which one is used by the HLQ or AuthID



## Creating Logical Transaction Stores

- **Replicate the transaction store tables**
  - CREATE TABLE EDIENU32.EDITSEV
  - CREATE TABLE EDIENU32.EDITSGP
  - CREATE TABLE EDIENU32.EDITSTU
  - CREATE TABLE EDIENU32.EDITSTH
  - CREATE TABLE EDIENU32.EDITSTI
  - CREATE TABLE EDIENU32.EDITSAU
  - CREATE TABLE EDIENU32.EDITSTO



## Configuring Applications to Use Logical Stores

- **Set the AuthID or HLQ appropriately on the database connection parameters**



## Archiving the Transaction Store

- TRANSACTION DATA EXTRACT command This command extracts detailed information about transactions, sorted by transaction handle. You can use this command to create report data to:
  - Report on the number of purchase orders sent in a given period of time
  - Report on the total number of bytes sent in a given period of time (this can be useful for charging back costs to other departments based on their EDI usage)
  - Create a customized functional acknowledgment tracking report by application or by department; for example, an exception report on purchase orders sent more than 2 days ago that have not been acknowledged
  - Create an exception report flagging missing control numbers for inbound envelopes You can also use this command for functions other than reporting, such as:
    - Archiving Transaction Store data
    - Loading status data directly into the application by application key; for example, the status for each invoice sent could be loaded into the billing system by invoice number



## Purging the Transaction Store

- **PURGE command**
- This command marks a transaction for purging from the Transaction Store but does not remove it.
- ACFIELD(starting application control field data) TO(ending application control field data)
- APPLID(application ID)
- APPRECID(application receiver department ID)
- APPSNDID(sender's department ID)
- BATCH(translated transaction batch ID)
- DIR(processing direction)
- DLVDATE(starting delivery date) TO(ending delivery date)
- DLVTIME(starting delivery time) TO(ending delivery time)
- ENVDATE(starting transaction envelope date) TO(ending transaction envelope date)
- ENVTIME(starting transaction envelope time) TO(ending transaction envelope time)
- ENVTYPE(transaction envelope type)
- EPURDATE(starting transaction purge date) TO(ending transaction purge date)
- FORMAT(data format ID)
- FUNACKP(pending functional acknowledgment)
- GRPCTLNO(starting sender's group control number) TO(ending sender's group control number)
- HANDLE(starting transaction ID) TO(ending transaction ID)
- IFCC(override condition codes)





## Purging the Transaction Store (cont.)

- INTCTLNO(starting sender's interchange control nbr) TO(ending sender's interchange control nbr)
- INTRECID(interchange receiver ID)
- INTSNDID(interchange sender ID)
- NETACKP(pending network acknowledgment)
- NETID(network ID)
- NETSTAT(network transaction status)
- SETCC(condition codes)
- SNDDATE(starting request sent date) TO(ending request sent date)
- SNDTIME(starting request sent time) TO(ending request sent time)
- STDTRID(EDI standard transaction set ID)
- STSTAT(transaction status)
- TPID(trading partner ID)
- TPNICKN(trading partner nickname) TRERLVL(maximum translation error level)
- TRXCTLNO(starting transaction set control number) TO(ending transaction set control number)
- TRXDATE(starting transaction date) TO(ending transaction date)
- TRXSTAT(transaction processing status)
- TRXTIME(starting transaction time) TO(ending transaction time)
- Example: Mark for purging all EDI documents that have been delivered to trading partner PISCES and accepted. `PERFORM PURGE WHERE TPNICKN(PISCES) TRXSTAT(61)`



## Pruning the Transaction Store

- **REMOVE TRANSACTIONS command** This command deletes transactions from the transaction store
- ACFIELD(*starting application control field data*) TO(*ending application control field data*)
- APPLID(*application ID*)
- APPRECID(*application receiver department ID*)
- APPSNDID(*sender's department ID*)
- BATCH(*translated transaction batch ID*)
- DIR(*processing direction*)
- DLVDATE(*starting delivery date*) TO(*ending delivery date*)
- DLVTIME(*starting delivery time*) TO(*ending delivery time*)
- ENVDATE(*starting transaction envelope date*) TO(*ending transaction envelope date*)
- ENVTIME(*starting transaction envelope time*) TO(*ending transaction envelope time*)
- ENVTYPE(*transaction envelope type*)
- EPURDATE(*starting transaction purge date*) TO(*ending transaction purge date*)
- FORMAT(*data format ID*)
- FUNACKP(*pending functional acknowledgment*)
- GRPCTLNO(*starting sender's group control number*) TO(*ending sender's group control number*)
- HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
- IFCC(*override condition codes*)
- INTCTLNO(*starting sender's interchange control nbr*) TO(*ending sender's interchange control nbr*)
- INTRECID(*interchange receiver ID*)



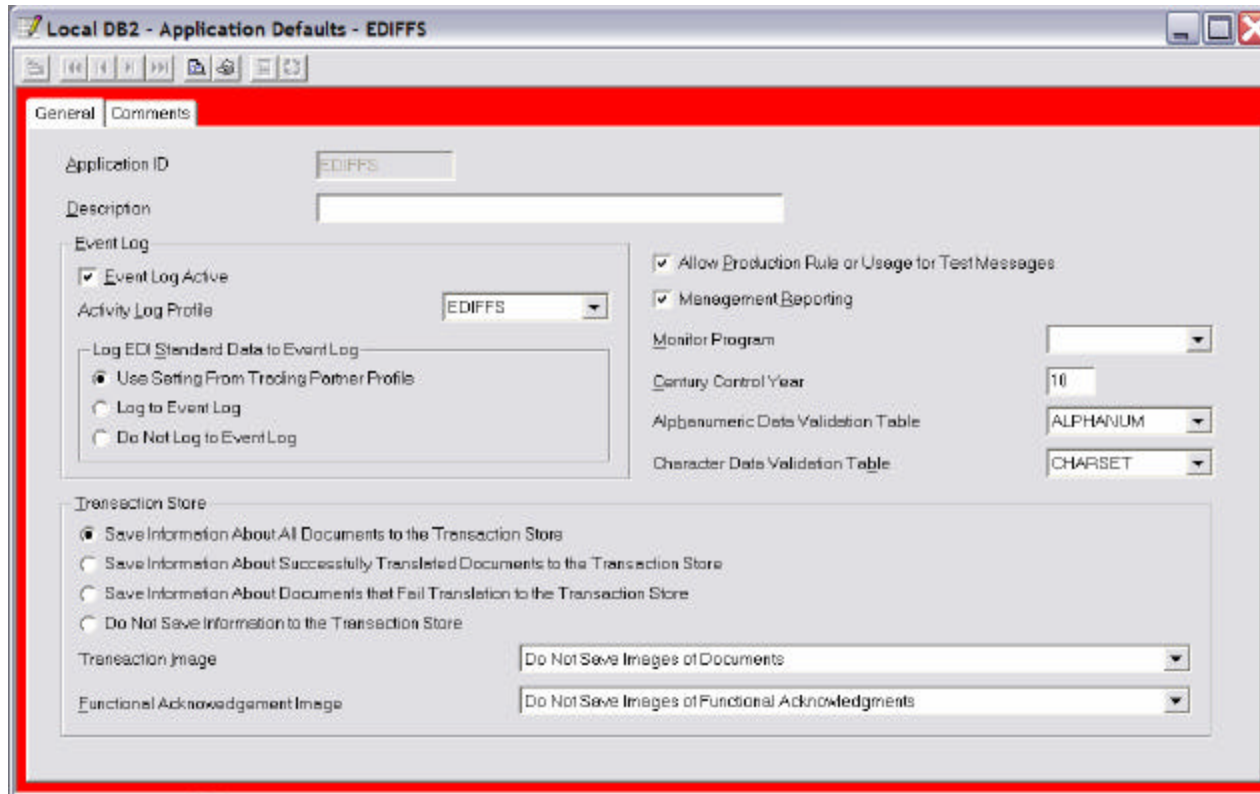
## Pruning the Transaction Store (cont.)

- INTSNDID(*interchange sender ID*)
- MAXRUNTIME(*maximum remove runtime minutes*)
- NETACKP(*pending network acknowledgment*)
- NETID(*network ID*) NETSTAT(*network transaction status*)
- NUMDELS(*number of database deletes before commit*)
- SETCC(*condition codes*)
- SNDDATE(*starting request sent date*) TO(*ending request sent date*)
- SNDTIME(*starting request sent time*) TO(*ending request sent time*)
- STANDALONE(*operate DataInterchange only*)
- STDTRID(*EDI standard transaction set ID*)
- STSTAT(*transaction status*)
- TPID(*trading partner ID*)
- TPNICKN(*trading partner nickname*)
- TRERLVL(*maximum translation error level*)
- TRXCTLNO(*starting transaction set control number*) TO(*ending transaction set control number*)
- TRXDATE(*starting transaction date*) TO(*ending transaction date*) TRXSTAT(*transaction processing status*)
- TRXTIME(*starting transaction time*) TO(*ending transaction time*)
- Example: Delete all eligible transactions that are more than 30 days old. Only transactions that have a status of PURGE-USER REQUEST or PURGE-DATE EXPIRED are eligible. PERFORM REMOVE TRANSACTIONS WHERE HANDLE(\*-999) TO(\*-30)

## Restoring Entries to the Transaction Store

- **WDI does not support restoring messages to the transaction store**
- **You can create a viewer tool for the TRANSACTION DATA EXTRACTs to handle the requirement for viewing archived transactions**
- **Another option is to create a parallel “archive” transaction store**
  - Move data to the archive stores to purge
  - Back up the archive stores and empty them
  - Restore to the archive store and view there

## Reducing the Volume of Data put to the Transaction Store



Local DB2 - Application Defaults - EDIFFS

General | Comments

Application ID: EDIFFS

Description:

Event Log

Event Log Active

Activity Log Profile: EDIFFS

Log EDI Standard Data to Event Log

Use Setting From Trading Partner Profile

Log to Event Log

Do Not Log to Event Log

Allow Production Rule or Usage for Test Messages

Management Reporting

Monitor Program:

Century Control Year: 10

Alphanumeric Data Validation Table: ALPHANUM

Character Data Validation Table: CHARSET

Transaction Store

Save Information About All Documents to the Transaction Store

Save Information About Successfully Translated Documents to the Transaction Store

Save Information About Documents that Fail Translation to the Transaction Store

Do Not Save Information to the Transaction Store

Transaction Image: Do Not Save Images of Documents

Functional Acknowledgement Image: Do Not Save Images of Functional Acknowledgments

## Securing the Transaction Store

- **Use the DB2 view security technique to create views that can only see certain logical logs**
- **CREATE VIEW EDIBUYER.EDIELOG AS SELECT \* FROM EDIEC32E.EDIELOG WHERE APPLID = 'PURCHSNG'**



## Correlating Events with the Transaction Store

- **On Windows the log can be correlated to the transaction store by matching on event log date and time with the transaction store created date and time**
- **On AIX and z/OS the log can be correlated to the transaction store by matching the transaction handle to the event ID**

## Operating without the Transaction Store

- **Pros**

- Can save time and improve throughput in environments that are “pushing the envelope” if turned off
- If transactions are logged else anyway, may be redundant

- **Cons**

- Can't do duplicate detection, FA reconciliation, overdue FA detection, delayed enveloping, etc.
- Can't do remote debugging through the client





## Best Practices Discussion

- **Separate Transaction Stores by application for best performance**
- **Limit messages stored to just the ones that are required**
- **Limit access using DB2 view technique**
- **Run periodic jobs to unload the Transaction Store**
- **Organize unload files by how they are searched**
- **Synchronize unloads with the event log**
- **If performance is not good enough, consider running without the Transaction Store**

## Best Practices Case Study

- **Nestle GLOBE**

- PURGEINT parameter is 40 days
- One APPLID=EDIFFS
- Store=Y, Images=Y, FuncAck Images=Y, Log EDI Std Data=Log to Event Log, Management Reporting=ON
- Archive daily to 2 files (sent & received) using PERFORM TRANSACTION DATA EXTRACT SELECTING INTERCHANGE(Y) GROUP(Y) TRANSACTION(Y) IMAGE(Y) WHERE FUNACKP(N) STSTAT(3) WHERE FUNACKP(N) STSTAT(4) DIR(S|R);
- PERFORM REMOVE executed in next step of archive job
- Statistics cleaned up using PERFORM REMOVE STATISTICS WHERE PRIORTO(\*-30) NUMDELS(25)



## Generating Alerts

How to get notified when something goes wrong



## Alerts

- **Ways of detecting when something goes wrong**
- **An architecture for alerting**
- **Best practices case study**



## Ways of Detecting When Something Goes Wrong

- **Condition codes**
- **API return codes**
- **Trigger exit programs**
- **Triggering off the PRTFILE Queue**
- **PRTFILE scanning**

## Using JCL Condition Codes to Trigger Alerts

- **Send/Receive Translation (S/R maps)**
  - 1 and 2 may be OK based on acceptable error level
  - 3-119 should be data related errors, WDI can continue
  - 120 and greater are serious problems, WDI is probably down
  - Use IFCC() SETCC() to filter uninteresting events like “no data in input file” (6)
- **Data Transformation (DT maps)**
  - 0, 4 are OK
  - 8 indicates a data related error, WDI can continue
  - 12 is a serious problem, WDI is probably down



## Using ediservr Return Codes to Trigger Alerts

- **ediservr return codes**
  - Value is 0, 8, or 12
  - 8 means data error, msg in exception status
  - 12 means severe error, WDI is now likely unable to process any messages

## Using API Return Codes to Trigger Alerts

- **Anything other than 0 or 5 is a severe API configuration error**
  - assume WDI is now unable to process any messages
- **5 is a utility error, use `GetRetCodes(rc, erc)`**
  - RC=8 means data error, msg in exception status
  - RC=12 means severe error, WDI is now likely unable to process any messages





## Using Trigger Exit Programs to Trigger Alerts

- `bool bProceed msgTransform(void* pvExitContext, long rc, long ccbrc, long ccberc)`
- Results of the transformation
  - contained in `rc`, `ccbrc` and `ccberc`
  - Meaning is the same as for the C++ API
- Return values
  - `SYNC_CONTINUE` - syncpoint and then continue processing.
  - `SYNC_TERM` - syncpoint and then terminate.
  - `CONTINUE` - roll back the current message and continue processing from the input queue.
  - `TERMINATE` - roll back the current message and terminate the adapter.

## Triggering Off the PRTFILE or Failure Queues

- **If `genprtfile=onerror` | blank in the `wdi.properties` file, then the PRTFILE will be sent to a queue**
- **You can trigger off of that queue**
- **The failure queue will also receive the bad message if an error occurs**
- **You can trigger off of the FAILURE queue as well**
- **Both provide the same set of trigger events**

## Using PRTFILE scanning to Trigger Alerts

- **When using DT maps**
  - The regex “FF0584 The PERFORM TRANSFORM command completed with a severity code of (08|12)” should do it
  
- **When using send/recv maps**
  - The regex “FF0162 Immediate error attempting to translate the next transaction, return code = (8|12)”

## An Architecture for Generating Alerts

- The user can generate Events when using WDI that can be handled by various Event handlers like EMail, Pager etc.
- The selection criteria used for generating these Events is based on Severity Threshold and/or Message ID's. The user can chose to generate events for any messages that have Severity greater than (0/4/8/12) or for any particular Message occurrence.
- The message ID selection takes precedence over Severity in case both are used for filtering. Message IDs can be selected to be include or excluded from the output.
- The output of this event generation can be a PRINT file/XML format Print file/ADF format print file/Event log. This output is generated at the end of a transformation.
- The Application defaults has entries for above 4 outputs, corres ponding criteria (Message ID's and Severity thresholds) and a destination profile entry.
- The output file generated is routed to a destination and this destination is defined in the Destination Profile. There could be multiple destination profiles defined in a system.
- The following are different type of destinations and the corresponding technique to pick the output file.
  - MQ Queue - A trigger program gets invoked for each event messages.
  - File Directory - A program waits on the directory and picks any files created in that directory.
  - Datasets/File - A program is invoked at the end of translation that processes t he file.
  - CICS TS Queue - A transaction is triggered that reads data out of the queue.



## An Architecture for Generating Alerts (cont.)

- The "Plugin Event handler" module handles the above reading of the output.
- Each of the destination profile has an Event Handler Profile associated with it.
- This Event handler profile refers to the actual handler program that processes the event output. It also has all properties required by the Handler as Key/Value Pairs. The user can define multiple Handler profiles.
- The "Plugin Event handler" module fetches data to the Handler program along with corresponding handler properties.
- Email handler program is one of these Event handlers that is provided with WDI.
- This Email handler program is written in Java using JavaMail API. It uses SMTP protocol to send Emails. This is an open source and the user is given a chance to customize the program.
- The user can add any other handler programs to WDI based on the requirements.
- The email programs send a message either to an error processor, the trading partner, or both.
- The email programs logs a the fact that it successfully sent a m essage, who it sent it to and for what failed message.
- If the error email message bounces, then send a message to the error processor in the host company.

## Best Practices Case Study

- **Lynn Draiss of New York State DMV**
- **JCL job step after WDI step**
  - count output documents produced and append to PRTFILE
  - Put the modified PRTFILE on an MQ queue to a Windows box
- **VB program on Windows box**
  - Scans PRTFILE and checks each message ID against a list of known message IDs
  - If a “notify” or unrecognized message ID comes up, then
    - creates a word doc describing the problem taking information from the messages
    - Send the word doc to the trading partner via their email address in the WDI Trading partner Profile
    - Log the Message ID, the email and the date/time and that it was sent
  - Checks the “from” mailbox for “bounce” messages and forwards them to an administrator for manual processing
- **Result is a huge reduction in manual processing of failed inbound messages**



## Replay/Resend

Reprocessing failed messages



## Replay/Resend

- **Use cases for replay/resend**
- **Tools for resending and replaying messages**
- **Best practices discussion**





## Use Cases for Replay and Resend

- **Replay**

- An inbound message failed translation because of an error in the map or the configuration
- Message was moved to exception status
- User has fixed the problem and wants to retry the msg

- **Resend**

- A trading partner claims they never received a msg that you sent to them
- User wants to resend the previously sent message

## Tools for Replay/Resend – Transaction Store

- **RECONSTRUCT command** - This command takes information that has been saved in the Transaction Store and rebuilds an interchange just as it was sent or received (using the same control numbers). This command can be used if your trading partner lost an interchange you sent, and you must send the same interchange again. It can also be used to rebuild interchanges sent to you.
- **Syntax**
  - RECONSTRUCT DIR(*processing direction*) FILEID(*processing file dname*) IFCC(*override condition codes*) INTCTLNO(*starting sender's interchange control nbr*) TO(*ending sender's interchange control nbr*) INTRECID(*interchange receiver ID*) INTSNDID(*interchange sender ID*) PAGE(*pageable translation*) RAWDATA(*translate to raw data format*) SETCC(*condition codes*) TPNICKN(*trading partner nickname*) RECONSTRUCT
- **Example** PERFORM RECONSTRUCT WHERE TPNICKN(MYTP) INTRECID(123456789) INTCTLNO(5) DIR(R) FILEID(AUDITOR)

## Tools for Replay/Resend – Transaction Store

- **REENVELOPE command** This command takes the EDI transactions from the Transaction Store that were previously enveloped, envelopes them again, and places the results in an envelope file.
- **Syntax REENVELOPE** ACFIELD(*starting application control field data*) TO(*ending application control field data*) APPLID(*application ID*) APPRECID(*application receiver department ID*) APPSNDID(*sender's department ID*) BATCH(*translated transaction batch ID*) DIERRFILTER(*initial error filter set*) ENVDATE(*starting transaction envelope date*) TO(*ending transaction envelope date*) ENVPRBREAK(*start new envelope*) ENVTIME(*starting transaction envelope time*) TO(*ending transaction envelope time*) ENVTYPE(*transaction envelope type*) EPURDATE(*starting transaction purge date*) TO(*ending transaction purge date*) FILEID(*processing file ddname*) FIXEDFILEID(*fixed-to-fixed output ddname*) FORMAT(*data format ID*) FUNACKP(*pending functional acknowledgment*) GRPCTLNO(*starting sender's group control number*) TO(*ending sender's group control number*) HANDLE(*starting transaction ID*) TO(*ending transaction ID*) IACCESS(*IEXIT access*) IAREA(*IEXIT information*) IEXIT(*interchange control program*) IFCC(*override condition codes*) INMEMTRANS(*transactions in memory*) INTCTLNO(*starting sender's interchange control nbr*) TO(*ending sender's interchange control nbr*) INTRECID(*interchange receiver ID*) INTSNDID(*interchange sender ID*) ITPBREAK(*new interchange envelope*) ITYPE(*IEXIT program type*) NETACKP(*pending network acknowledgment*) NETID(*network ID*) NETSTAT(*network transaction status*) OPTRECS(*optional record type*) OUTFILE(*output data file name for SAP Status*) OUTTYPE(*output data file type for SAP Status*) PAGE(*pageable translation*) RAWDATA(*translate to raw data format*) RECOVERY(*recovery unit of work*) SAPUPDT(*track SAP status*) SNDDATE(*starting request sent date*) TO(*ending request sent date*) SNDTIME(*starting request sent time*) TO(*ending request sent time*) SERVICESEGVAL(*service segment validation level*) SETCC(*condition codes*) STDTRID(*EDI standard transaction set ID*) TPID(*trading partner ID*) TPNICKN(*trading partner nickname*) TRERLVL(*maximum translation error level*) TRXCTLNO(*starting transaction set control number*) TO(*ending transaction set control number*) TRXDATE(*starting transaction date*) TO(*ending transaction date*) TRXSTAT(*transaction processing status*) TRXTIME(*starting transaction time*) TO(*ending transaction time*) VERIFY(*verify transaction status*) REENVELOPE
- Example
  - PERFORM REENVELOPE WHERE HANDLE(2001121410153300001) TRXSTAT(31)

## Tools for Replay – MQ Adapter Based

- **WDI adapter**
  - Saves input messages as files in the rcv directory
  - Files are named MQ\_Msg\_ID.rcv
  - These files can be requeued using standard MQ tools like rfhutil
  
- **Multi-threaded adapter**
  - Saves input messages that fail to the failure, or back out queue
  - Messages can be requeued from the failure queue using standard MQ tools



## Tools for Replay JCL and Command Line Based

- **Failed input messages are written to the exception file**
- **For DT maps all input msg types go to the exception file**
- **For Send/Recv maps**
  - Data Formats, only the failing document is written to the exception file
  - For EDI nothing is written to the exception file
- **For replay manually retranslate the exception file**



## Best Practices Discussion

- **For enveloped data types like EDI, smaller messages with fewer included documents (ideally just one) are going to have fewer issues associated with reprocessing them**
- **If you must handle inbound envelopes with multiple documents, use envelope level recovery if possible so that you can reprocess entire input envelopes**
- **If you can't use envelope recovery, then avoid replay if possible and have sender resend instead**



## Dealing with Overdue Acknowledgments

Handling the situation when there is no answer at the other end



## Overdue Acknowledgments

- **Overdue acknowledgment use cases**
- **Tools for detecting overdue functional acknowledgments**
- **Best practices discussion**
- **Best practices case study**



## Overdue Acknowledgment Use Cases

- **Real time use case**
  - If a message isn't acknowledged within 15 minutes then resend the message
  - If the message is resent 3 times and still no acknowledgment, then fail the message and raise an alert
- **Batch use case**
  - Once a day process all the received functional acknowledgments
  - Format a report detailing any unacknowledged messages more than 3 days old



## Tools for Detecting Overdue Functional Acknowledgments

- **TRANSACTION DATA EXTRACT command** This command extracts detailed information about transactions, sorted by transaction handle
- **Syntax** TRANSACTION DATA EXTRACT ACFIELD(*starting application control field data*) TO(*ending application control field data*) APPLICATION(*write application data record*) APPLID(*application ID*) APPRECID(*application receiver department ID*) APPSNDID(*sender's department ID*) BATCH(*translated transaction batch ID*) CONCATENATE(*concatenate extract data*) DIR(*processing direction*) DLVDATE(*starting delivery date*) TO(*ending delivery date*) DLVTIME(*starting delivery time*) TO(*ending delivery time*) ENVDATE(*starting transaction envelope date*) TO(*ending transaction envelope date*) ENVTIME(*starting transaction envelope time*) TO(*ending transaction envelope time*) ENVTYPE(*transaction envelope type*) EPURDATE(*starting transaction purge date*) TO(*ending transaction purge date*) FORMAT(*data format ID*) FUNACKP(*pending functional acknowledgment*) GROUP(*write group data record*) GRPCTLNO(*starting sender's group control number*) TO(*ending sender's group control number*) HANDLE(*starting transaction ID*) TO(*ending transaction ID*) IFCC(*override condition codes*) IMAGE(*write image data record*) INTCTLNO(*starting sender's interchange control nbr*) TO(*ending sender's interchange control nbr*) INTERCHANGE(*write interchange data record*) INTRECID(*interchange receiver ID*) INTSNDID(*interchange sender ID*) NETACKP(*pending network acknowledgment*) NETID(*network ID*) NETSTAT(*network transaction status*) RECEIVEACKDATA(*write detailed acknowledgment data*) RECEIVEACKIMAGE(*write receive acknowledgment record*) SENDACKDATA(*write detailed acknowledgment data*) SENDACKIMAGE(*write acknowledgment record*) SETCC(*condition codes*) SNDDATE(*starting request sent date*) TO(*ending request sent date*) SNDTIME(*starting request sent time*) TO(*ending request sent time*) STDTRID(*EDI standard transaction set ID*) STSTAT(*transaction status*) TPID(*trading partner ID*) TPNICKN(*trading partner nickname*) TRANSACTION(*write transaction data record*) TRERLVL(*maximum translation error level*) TRXCTLNO(*starting transaction set control number*) TO(*ending transaction set control number*) TRXDATE(*starting transaction date*) TO(*ending transaction date*) TRXSTAT(*transaction processing status*) TRXTIME(*starting transaction time*) TO(*ending transaction time*) USERPGM(*user program*) TRANSACTION DATA EXTRACT
- Example: PERFORM TRANSACTION DATA EXTRACT SELECTING INTERCHANGE(Y) GROUP(Y) TRANSACTION(Y) WHERE FUNACKP(Y) ENVDATE(01/01/01) TO(\*-3)

## Best Practices Case Study

- **Jeff Pesick, Australian Customs**
  - They have created a custom logging program that
    - Records outbound messages
    - Matches inbound acknowledgments to the outbound messages
    - Periodically checks for unacknowledged messages and resends ones for which acknowledgments are overdue
    - After several resends it notifies operations via an email message