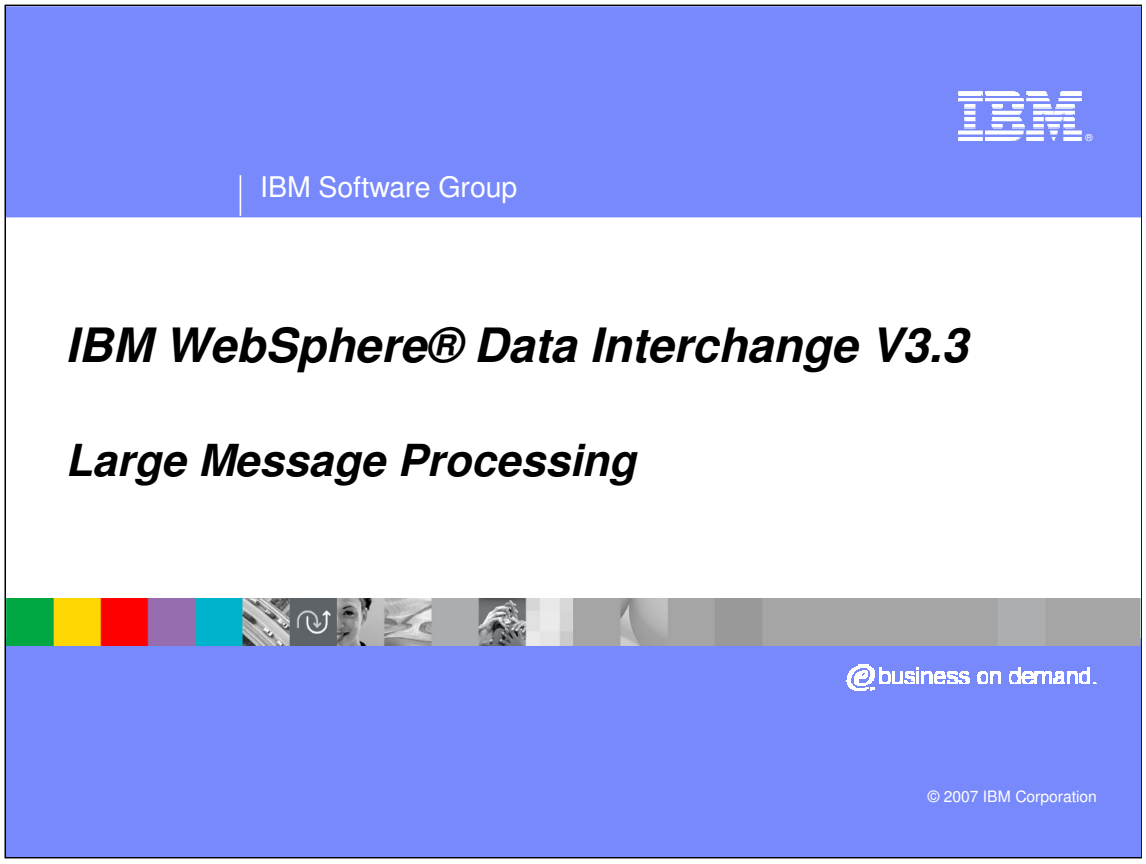


The presentation will review WDI processing options for large message processing.

Agenda

- Parse from file
- Pageable AMM
- XML Split
- EDIWORK and Output paging
- Pageable Translation for Send / Receive
- Limiting trace and audit file output
- Internal Performance Features
- “Best Practices” recommendations

The ability to process very large transactions (2 GB or so) or very large messages (hundreds of transactions) is an accepted requirement for WebSphere Data Interchange. Processing telephone bills for large companies with many locations, or all trust accounts handled by a bank supporting a Fortune 50 company or government entity can lead to this need. The problem is represented by two basic types: a) a single large transaction, or an interchange or message with a large number of transactions in the same recovery scope. The hardware resource most exposed with these data conditions is main storage or memory.

WebSphere Data Interchange has a number of techniques and features available to assist in the processing of very large messages, that is greater than 200 MB.

The first of these, “Parse from file”, allows a customer with a single large EDI interchange or EDI transaction to bypass parsing into memory and parse data directly from the file.

The pageable AMM feature, which pages the WDI Abstract Message, is always active, but can be controlled from the PERFORM command.

Feature three, XML Split, was created to allow a one-to-many output from a large XML document.

Output paging for DT maps allows the user to tell WDI to write output to a file rather than save in in memory for the duration of translation.

Typical WDI processing

- WDI Utility reads the input
- LMA determines the message or interchange
- Message Broker is invoked
- Deenveloper invokes the parser to find a transaction
- WDI Nodes process the data
- Translated output, print file messages, transaction store records are all saved in memory
- Control returns to the Message Broker
- The next transaction is processed adding to data stored
- At the end of the interchange, control goes back to the Utility where all data is written to files
- LMA then finds the next message
- Process repeats until all interchanges in the input are processed

The WDI Utility reads the input file. The Logical Message Adapter identifies the input unit of work, e.g. an EDI Interchange, and passes it to the Message Broker. The Message Broker does a "lazy parsing" of the Interchange to get Envelope Information and the ST/SE data. This is passed to the Deenvelope component and subsequent components for processing. The output from translation, the audit file / print file messages, and transaction store data are all stored in link lists. Control is then returned back to the Message Broker, and the process is repeated for the next ST/SE, until all transactions are processed.

After all transactions are processed, the output link lists are then returned to the Utility. The output files are updated, the print file messages are written, and the Document Store tables are updated. The Logical Message Adapter then gets the next input unit of work, and repeats the process until the file is exhausted.

Parse from File

- Feature is triggered by a PERFORM keyword
- Using this feature can significantly reduce memory requirements
- Keyword on PERFORM TRANSFORM is PARSEFILE(Y)
- PARSEFILE only available for XML in WDI v3.2, SYNTAX(X), but valid for all syntaxes, SYNTAX(E) and SYNTAX(D), beginning with WDI v3.3
- Keyword INFILE specifies the input file logical name
- XML is restricted to having one and only one XML document in the input
- Not valid for WMQ as an input
- Not valid in CICS, and is ignored if specified
- Not valid when used with XMLSPLIT

A PERFORM TRANSFORM keyword named, PARSEFILE, will allow data to be parsed directly from the input file if the input file contains one logical message. Using this keyword can significantly reduce the memory needed for processing a large input file. Without this feature, WDI reads the entire document into a buffer, so if a 500 MB XML document is being processed, then WDI will use at least 500 MB to parse the data. With this option, the WDI parsing reads from the file and requires much less memory.

PARSEFILE(Y) means parse-from-file. The default is PARSEFILE(N), which means parse-from-buffer. PARSEFILE is only valid for XML input data in WDI version 3.2.1, and is not valid in CICS. PARSEFILE may be used with SYNTAX(E) and SYNTAX(D) in WDI version 3.3. If PARSEFILE(Y) is specified on a PERFORM TRANSFORM command, INFILE must exist and contain the name of the input file, and INTYPE must not have a value.

There is a restriction when using PARSEFILE(Y). The XMLSPLIT capability is not supported. This means that PARSEFILE(Y) should only be specified when the input file contains one and only one XML document. More than one document in the file will result in a fatal error in the parser.

Pageable AMM

- Applies only to DT maps, similar to Pageable Translation of S/R translator
- Controlled by the PAGETHRESHOLD keyword on a PERFORM TRANSFORM
- By default always “ON”, can be disabled with PAGETHRESHOLD(0)
- Allows memory allocated to the AMM to be limited
- Paging of subtree, repeating loops is automatic
- Paged data is stored in required file EDIPAGE; if not defined, PAMM is disabled
- Not valid for EDI transactions with Hierarchical Loops (HL segment) in WDI v3.2.1; restriction lifted in WDI v3.3

The "Pageable AMM" (PAMM) is similar to the "paging subsystem" called Pageable Translation in the Send / Receive Translator. This technique "pages out" or writes data that normally resides in memory to a file. Doing so reduces the memory requirement for large EDI translations. The Abstract Message Model or AMM is a WDI internal representation of data during translation. The PAMM will "page out" AMM nodes that are part of repeating data. When there is a repeating loop, element, segment, structure, or record, the AMM subtree containing the repeating data will be paged out of memory when the number of repetitions exceeds a given threshold and the data is no longer immediately needed. The default threshold is 1000. This threshold may be overridden using the PAGETHRESHOLD keyword. PAGETHRESHOLD(0) turns paging off.

The page file is named EDIPAGE. The file must be allocated or the Pageable AMM feature will not be used. If EDIPAGE is not defined, severity-0 message AM0016 will be written to the Print File and processing would continue normally. If an error occurs opening the page file, transformation would continue normally, though severity-0 message AM0015 would be logged. If an error occurs writing to the page file, transformation would continue normally, though severity-4 message AM0015 would be logged.

The EDIPAGE file contains the internal representation of both the input and the output data. Typically, you should allow several times the total of the input and (expected) output file sizes when determining how much space to allocate. This allows for the additional information (name, data type, parent-child pointers, etc.) that is kept in the file. The actual ratio depends on the document structure and the PAGETHRESHOLD value. For relatively verbose formats such as XML, the ratio will usually be lower, around 2-3 times the data size. For more condensed formats such as EDI, the ratio is likely to be higher, such as 3-5 times. If the PAGETHRESHOLD is higher, more of each document is kept in memory before starting to write data to the EDIPAGE file, so the EDIPAGE file will be smaller.

Note: The Pageable AMM function is primarily intended for reducing the memory used to process large individual documents (EDI transactions, Data Format transactions, and XML documents). Although it may slightly reduce memory usage when processing large EDI interchanges or other files with many medium sized transactions (2-10 MB), it may not be worth allocating the large page file produced in this case since the EDIPAGE file accumulates from one transaction to the next, until the end of the input file is reached. Increasing the PAGETHRESHOLD in this case can significantly reduce the size of the page file in this case, and keep more of each transaction in memory.

EDIPAGE may be allocated with any technique the user prefers.

To execute WDI without the PAMM feature, issue

PERFORM TRANSFORM WHERE ..... PAGETHRESHOLD(0)

To execute WDI and reduce the memory used during transformation, i.e. start using the PAMM sooner, issue

PERFORM TRANSFORM WHERE ..... PAGETHRESHOLD(500)

To execute WDI and delay using the PAMM, utilize

PERFORM TRANSFORM WHERE ..... PAGETHRESHOLD(1500)

A typical z/OS DD statement to allocate the EDIPAGE dataset

//EDIPAGE DD DSN=&&tempfile,UNIT=SYSDA,SPACE=(CYL,XXX)

A typical Windows command file statement to allocate the EDIPAGE file

set file(EDIPAGE,..\..\..\edipage.text);

Restrictions: The Pageable AMM feature is not available when using WDI inside CICS. The primary reason for this is the use of a sequential

Internal Performance Features

Internal features to improve WDI performance are:

1) WDI has a caching subsystem for objects such as Control Strings, Rules, and Trading Partners. This subsystem allowed subsequent calls to DB2 to be eliminated as long as the same object was being requested. Up to 5 instances of an object can be cached in a PERFORM cycle

2) On z/OS, a single the DB Connection request is made rather than in each logical node, this reduced the calls to the STEPLIB and DSNLOAD from 222,000 to 46. Each DB Connect accesses the Call Attachment Facility module (CAF).

3) WDI has a Node memory pool, so that the AMM does not have to issue "new" and "free" memory acquisition instructions, but can reuse existing, acquired storage.

Internal performance features incorporated into WDI are shown on this slide and the next.

Internal Performance Features

4) WDI reuses Parser instances; instead of creating a new instance of a parser for each time a parser is used, a previously initialized, loaded parser is used

5) WDI avoids repeatedly loading DLL's. WDI does not destroy the Message Flow until the WDI message broker is terminated.

6) WDI provides an ability to suppress repetitive messages from the print file (RU0003, UT0008, FF0007); suppression is based on the attribute VERBOSE_NO;

7) WDI avoids repeated allocation of output buffer by keeping the output buffer intact between calls.

8) For large message processing, if the parsing work file is allocated and the input file contains multiple messages, the logical message adapter will begin using the parsing work file to hold the logical message. This replaces the buffer allocation to hold the logical message.

XML Split Logic

- XML SPLIT(Y) splits a single XML document into smaller individual documents, creating the ability to do one-to-many processing for XML
- Split points are identified by XML tags in the XML General tab of an XML schema or DTD
- WDI saves Headers and Trailers to be wrapped around the repetitive data to create “internal transactions” to be processed
- An XMLWORK file is required and used to construct the internal messages
- The header element tag must be at loop level 1

The WDI XML logical message adapter (LMA) allows splitting of a single XML document into smaller individual documents. This provides a one-to-many capability for large XML documents with repeating groups. Using this feature greatly reduces the amount of memory used to process a very large XML document, but it is a special case use.

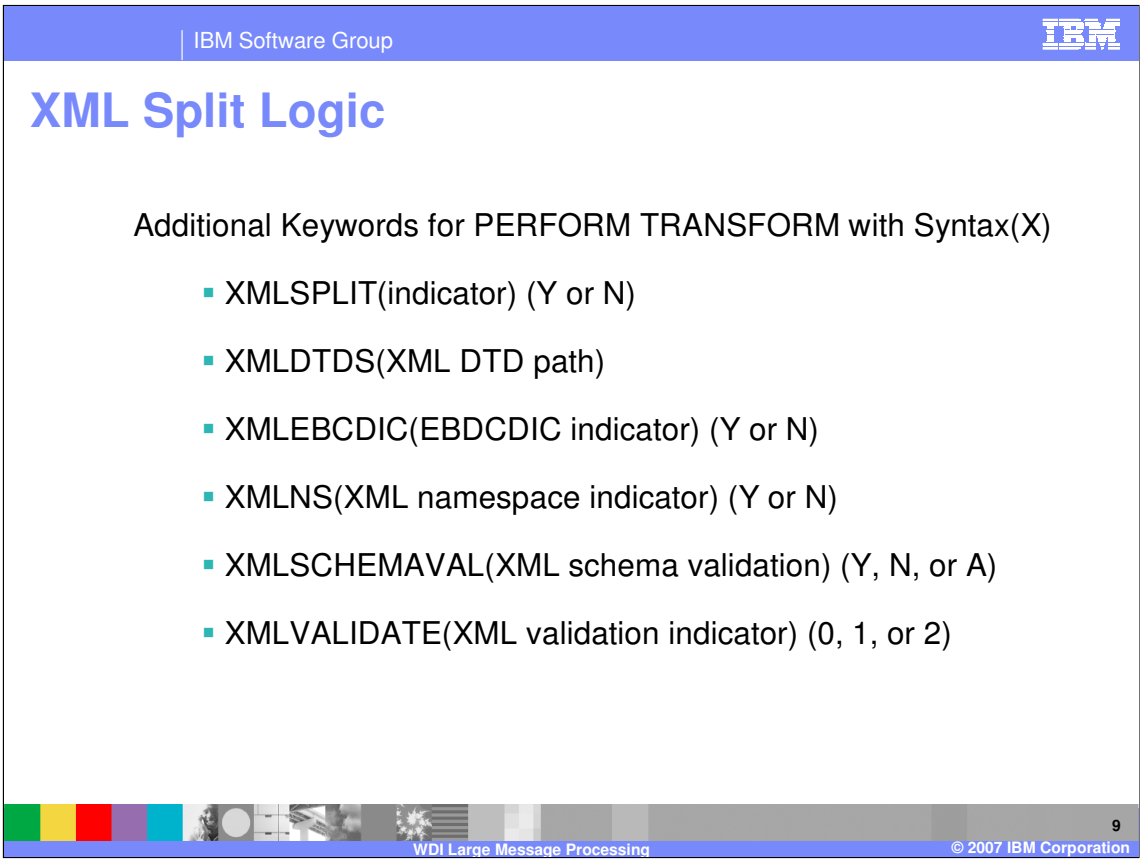
When XMLSPLIT(Y) is specified, the LMA will go through the XML LMA to get the beginning/ending of the XML document. New functions are called to split the XML document using these 2 pointers. The functions save the XML Header and Trailer information in a Virtual Array (VA) to reconstruct the split XML documents. WDI uses tag names to identify how to split the XML document - the tag names are specified on the XML General tab of WDI Client.

The first header area is the beginning of the XML document starting with <?XML up to the Header Element tag. For each Header Element tag in the input file, the data will be replaced in the Header VA without replacing the <?XML portion of the VA. The split messages from Message Element up to Message Element end tag will be written to the XMLWORK file. The data at Trailer Element tag up to the next Header Element tag - will be placed in the trailer VA. If the Trailer Element tag is not equal to the Root end tag, the root end tag is moved to the trailer VA. This tag should be the tag immediately following all the split XML documents. Header Element is optional. There is still a Header area VA but it contains the beginning of the document up to the first Message Element tag.

After the header/trailer have been located, the XMLWORK file is opened and read to construct the split XML documents. These are placed in a message VA as needed. The Header VA data is moved to offset 0 with each XML split message so the start of the message should always be at offset 0.

WDI will scan through the message VA looking for the next XML tag specified in the Message Element tag to identify the Detail area. This returns Detail Start/Detail End or an indicator that WDI is finished with the current XML document or needs to process a new XML document. After the XMLWORK file has been processed, the LMA will return an indicator to read more data from the original input file and begin the process again, searching for the next header/message/detail areas in the document.

This Header Element tag MUST be at a looping level of 1 (NO NESTED LOOPS) or unpredictable results will occur. WDI then constructs the XML document using the Header/Detail/Trailer virtual arrays and passes this back to the DT Utility in a buffer.



Keywords for PERFORM TRANSFORM with Syntax(X)

XMLSPLIT(indicator) (Y or N)
XMLDTDS(XML DTD path)
XMLEBCDIC(EBDCDIC indicator) (Y or N)
XMLNS(XML namespace indicator) (Y or N)
XMLSCHEMAVAL(XML schema validation) (Y, N, or A)
XMLVALIDATE(XML validation indicator) (0, 1, or 2)

The XML Split feature is available for XML Schemas and XML DTDs. To identify the XML tags used during the splitting process, use the Overview tab on the XML Schema. A right mouse click allow you to access a drop down list that populates the XML General tab with the tag names. The XMLSPLIT keyword must be set to Y on the PERFORM TRANSFORM to sue XMLSPLIT. This is a default.

XML Data Structure for splitting

```
?xml
Root element
header 1 element
header 2 element
...
/header n end
Tag1 Loop start (service provider)                                  document header
        tag2
        tag3
                tag4
                tag5
        tag6
                tag7 loop start (natuurlikpersen)                   message header
                        tag8
                        tag9
                        tag10 loop start
                                tag 11
                                                                    tag 12
                                tag 13
                        /tag 10 loop end
                        tag14
                /tag7 loop end
                tag 15
        /tag6 end                                                                   message trailer
        tag16
/tag 1 loop end
trailer 1                                                                                   document trailer
trailer 2
...
/trailer n end
/root end
```


An example of a data structure that can be used with XMLSPLIT is shown above. The constructs in red are not supported

XML Split Example

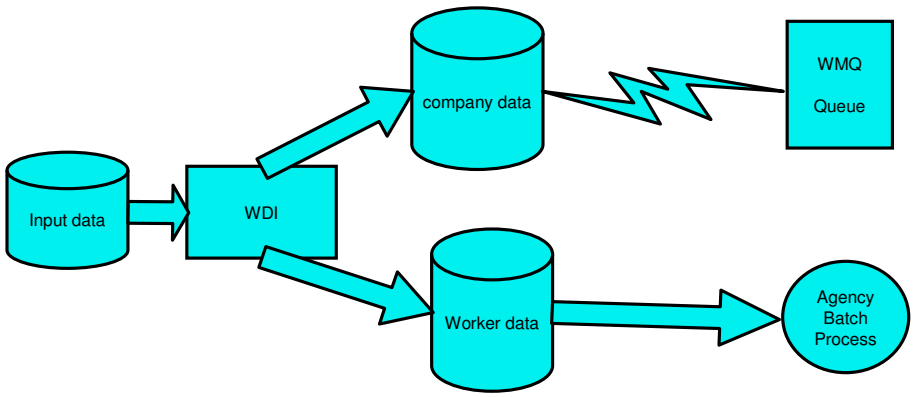
A government agency receives a single XML transaction with one iteration of a loop for each employee in an organization. Some organizations have several million employees on which they report in this transaction. The XML transaction, defined in a Schema, can be 2 GB or more.

The agency requires that the single XML transaction be translated into one XML transaction for each employee, and a second, summary XML transaction be created for organization totals.

The solution is to be run on an AIX machine, using file input. The employee XML transactions will be output to a file, and the summary organization XML transaction is to be sent to a WMQ Queue.

This is the scenario for an example which XMLSPLIT is required.

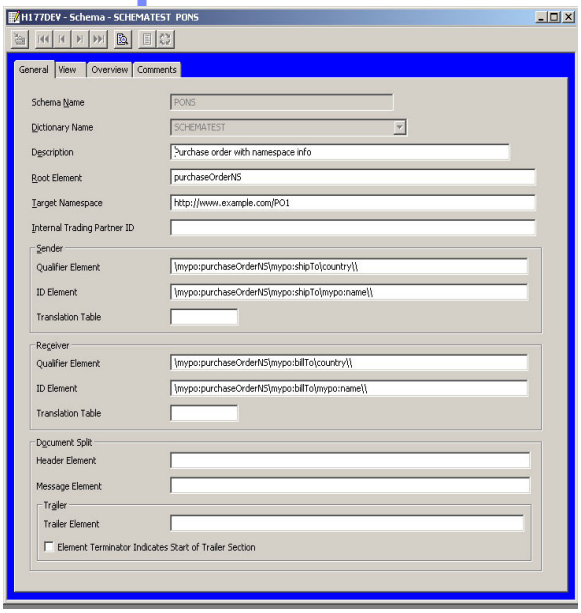
Agency Flow



In this example, the input data is processed by WDI and split into company data and worker data. Those individual data outputs can be routed as needed.

Setting up for XML Split

- Within the Document Split
  - Define the Tag for the end of the header portion of the XML transaction
  - Define the Repeating area by Tab
  - Define the Tag starting “trailer” portion
  - Use a right mouse click on the Overview tab to place the tag names in the general tab fields



Define the fields as indicated on the XML General tab. The fields are easily populated from the Overview tab using a right mouse click on the desired tag, and then specifying the general tab field into which the tag name should be placed.

The “header portion” starts with the start of the XML document and continues until the tag name specified in the Header Element field. The repeating portion is specified by the tag name of the “message element” field. The “trailer portion” is specified by putting the tag name of the first element in the trailer, or by specifying the last tag name in the repeating element structure and checking the box at the end. This indicates that there are no segments after the repeating portion.

EDIWORK and output paging

- Pages output between transactions of a large interchange
- Available for DT maps with all syntax types
- Activated by keyword PAGE(Y) on PERFORM TRANSFORM command
- Has an internal threshold before workfile is used
  - A single transaction of 10MB or more, or,
  - A collection of smaller documents with a memory usage of 10MB
- Requires EDIWORK file to be allocated to hold output, print messages, and transaction store output
  - If not allocated, then a warning message is issued and processing continues as if PAGE(N)
  - If an error occurs writing to EDIWORK (no space), then the translation is terminated abnormally
- Not valid in CICS

The purpose of this capability is to reduce the amount of memory used during DT translations by "paging out" output data to a work file. The PAGE(Y) keyword on PERFORM TRANSFORM will cause output data to be paged to a work file named EDIWORK once a size threshold is met.
With PAGE(N), output data is collected in a buffer in the Message Broker as translation takes place. When the translation is complete, the output data buffer is passed back to the WDI Utility component where it is written to the user specified output file, as before.

With PAGE(Y) the Message Broker will write the output data to a work file after a size threshold is met. The workfile offset is passed back to the Utility, where the data is read from the work file, and written to the user specified output file, as before.

There are two places in the Message Broker that check the output size threshold in order to possibly page the output. If a single output document reaches 10 MB, then the document will be released from memory and written to the work file. If a collection of smaller output documents reach 10 MB, then any remaining output documents are written to the work file.

If PAGE(Y) is specified on a PERFORM TRANSFORM command and file EDIWORK is not defined, not allocated, or if there is a problem opening the file, a severity-4 message, MB0003 or MB0004, will be logged in the print file. In this case, processing would continue as if PAGE(N) was specified. If an error occurs writing to EDIWORK, then a severity-8 message, MB0005, will be logged and the translation will terminate.

The type of output data that may potentially be paged is not restricted. PAGE(Y) applies to ADF, EDI, and XML output data. There is a size restriction on the amount of data that can be paged to EDIWORK. The current restriction is 2 GB on z/OS. If this size is exceeded, a severity-8 message, MB0005, will be logged and the translation will terminate.

On z/OS, because EDIWORK may be reused each time the Utility calls the Message Broker (which occurs multiple times during a PERFORM command), EDIWORK cannot be defined as DISP=MOD or as an append-to-end-of-file file. The base EDIWORK offset is reset to zero each time the Utility calls the Message Broker.

PAGE(Y) is not valid in CICS on PERFORM TRANSFORM commands.

Pageable translation for Send Receive

- Triggered by using the keyword PAGE(Y) on a
  - PERFORM DEENVELOPE command
  - PERFORM DEENVELOPE AND TRANSLATE command
  - PERFORM TRANSLATE AND ENVELOPE command
  - PERFORM ENVELOPE command
  - PERFORM ENVELOPE AND SEND command
  - And others
- Can be activated by the TRCB VAXFLAG in the translator API
- Requires the EDIVAX dataset be allocated
- Has an internal threshold value before initiating, 28MB
- Not valid in CICS or MP, only available in z/OS batch
- Allows a transaction up to 2 GB to be processed

Pageable translation is designed to better utilize system memory during translation by transferring incoming data buffers to DASD once the allotted number (1000) of internal buffers has been exhausted. The maximum buffer size is 28,632 bytes. This means that approximately 28 MB of virtual storage can be used to hold data before pageable translation is triggered.

The PAGE keyword indicates whether pageable translation should be enabled. Valid values for Pageable Translation are: **Y** Enables Pageable Translation, and. **N** Disables Pageable Translation This keyword is used with the following commands:

DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
ENVELOPE AND SEND
RECEIVE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
RECONSTRUCT AND SEND
REENVELOPE
REENVELOPE AND SEND
RETRANSLATE TO APPLICATION
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO APPLICATION
TRANSLATE TO STANDARD

EDIVAX is the name of the pageable translation work file. This temporary work file is used by WebSphere Data Interchange when pageable translation is enabled and virtual storage usage for EDI or application data reaches 28 MB. For API applications, setting the VAXFLAG field in the TRCB to **X** enables pageable translation.

Limiting trace and audit file output

- FILTERMSGS – suppress a specified message from being written during translation
- IGNOREINFO – suppress all informational (severity 0) messages
- IGNOREWARN – suppress all warning level (severity 4) messages
- TRACELEVEL – suppress or limit the amount of trace entries collected during execution of a PERFORM TRANSFORM

Several keywords are available on the PERFORM TRANSFORM command to aid in reducing the amount of printed output (and its associated storage) during translation. These keywords are:

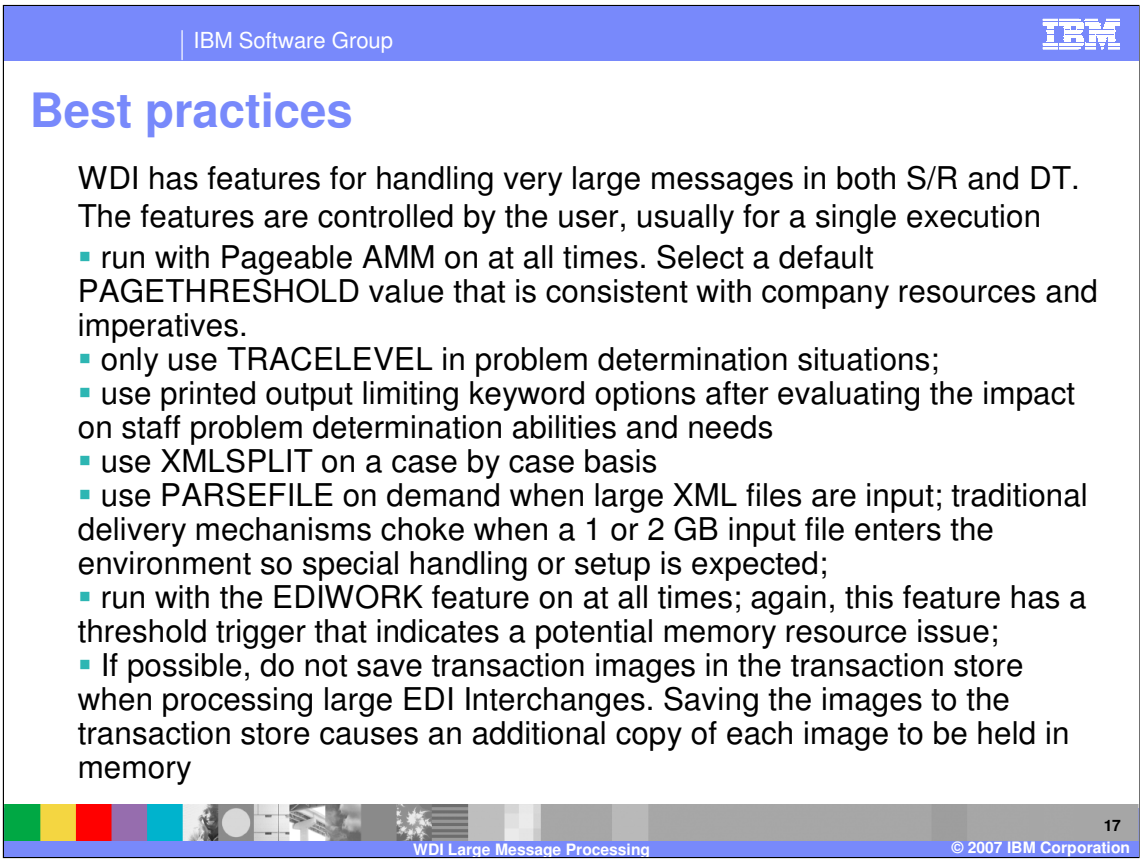
FILTERMSGS
IGNOREINFO
IGNOREWARN
TRACELEVEL

*FILTERMSG*S allows the specification of up to 10 WDI message Ids. For each ID specified, the corresponding message text is not written to the print file. With an interchange with hundreds of transactions this means selected informational messages or specific error messages can be suppressed.
For example,
PERFORM TRANSFORM
WHERE INFILE(ADF_IN) ... OUTFILE(EDIOUT) ...
FILTERMSGS(RU0003 UT0008)
SYNTAX(D);

IGNOREINFO is a Y or N keyword that will suppress all informational level (type I) messages. These might be the name of the file, rule selected, PERFORM command, etc. For interchanges with a number of transactions, this can greatly reduce the number of messages printed. It is both a memory saver and a CPU consumption reducer.

IGNOREWARN is a Y or N keyword that will suppress all warning level (type W) messages. It has a similar impact as IGNOREINFO.

TRACELEVEL(xy) where x is component and y is detail level of trace. This option should not be used except in problem determination situations. It has a large impact on processing time and resources when used at its most comprehensive levels. TRACELEVEL(C1) produces a fairly minimal trace in EDIDTTRC that shows heap storage in use at various points during DT processing. If a customer wants to use the various features discussed to effect memory usage, TRACELEVEL(C1) can be used as a tool to monitor such effects.



WDI has features for handling very large messages during both S/R and DT translations. The features are generally controlled by the user, usually for a single execution or setup.

As a "Best Practice" recommendation, the user should consider

-- running with Pageable AMM on at all times. The dynamic initiation of PAMM logic based on message size means smaller messages will be processed as usual, but large messages will invoke the special handling techniques in a "dark room" environment. Select a default PAGETHRESHOLD value that is consistent with company resources and imperatives. Increasing the PAGETHRESHOLD may be particularly useful if processing large EDI interchanges that contain many small to medium sized transactions.

-- only using TRACELEVEL in problem determination situations;

-- using printed output limiting keyword options after evaluating the impact of messages on staff problem determination abilities and needs and frequency of message need

-- using XMLSPLIT on a case by case basis; while XMLSPLIT is on by default, the feature is activated with the "split tags" specified with the XML DTD or schema.

-- using PARSEFILE on demand when large XML files are input; while WDI can process the file, traditional delivery mechanisms choke when a 1 or 2 GB input file enters the environment; it is anticipated that a message of this size will need special handling to make the file available to WDI

running with the EDIWORK feature on at all times; again, this feature has a threshold trigger that indicates a potential memory resource issue; using the feature as a default prevents unforeseen issues.

-- If possible, do not save transaction images in the transaction store - at least when processing large EDI Interchanges. Saving the images to the transaction store causes an additional copy of each transaction image to be held in memory during the DT processing.

Template Revision: 04/25/2006 11:09 AM

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	CICS	IMS	WMQ	Tivoli
IBM(logo)	Cloudscape	Informix	OS/390	WebSphere
e(logo)business	DB2	iSeries	OS/400	xSeries
AIX	DB2 Universal Database	Lotus	pSeries	zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

Other company, product and service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.